Oliviu Tatarov

Licenciatura em Ciências de
Engenharia e Eletrotécnica de Computadores

The application of Blockchain technology in the Insurance sector

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: João Paulo Pimentão, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente: Doutor Nuno Filipe Veríssimo Paulino – FCT NOVA

Arguente: Doutor João Almeida das Rosas – FCT NOVA

Vogal: Doutor Luís Filipe Lourenço Bernardo – FCT NOVA

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2020

The application of Blockchain technology in the Insurance sector

Agradecimentos

Quero começar por agradecer ao professor João Paulo Pimentão pela proposta de realização deste tema de tese. Agradeço à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa pelo conhecimento fornecido ao longo dos 5 anos e aos amigos que fui conhecendo ao longo do percurso que me ajudaram a chegar aqui.

Agradeço aos meus pais pelo apoio psicológico e por me financiarem os estudos.

Resumo

As fraudes, no setor financeiro, são as que possuem o maior impacto, em termos de perdas financeiras. Estas provêm de gerências, com pouca transparência e desrespeito para com a integridade da informação a que um cliente tem acesso. Estas práticas afetam o normal funcionamento e crescimento da indústria financeira no seu dia a dia, em específico a indústria das seguradoras. A possibilidade da alteração da apólice do cliente, sem o seu conhecimento ou consentimento, torna-o impotente perante situações em que possa vir a necessitar dos serviços que assegurou. Posto isto, é necessário um sistema que garanta integridade, confidencialidade, transparência, e segurança para os clientes. O *blockchain* é uma plataforma que oferece todas essas características.

Nesta tese fez-se o estudo e a seleção da plataforma de *blockchain* mais adequada, para completar as necessidades que a indústria das seguradoras tanto precisa. Depois de escolher a plataforma, prosseguiu-se com o desenvolvimento de um sistema, que servisse de prova de conceito em como a plataforma de *blockchain* oferece integridade, segurança, confidencialidade e transparência.

Palavras-chave: *Blockchain*, fraude, integridade, seguradoras.

Abstract

Fraud at the financial sector is, by far, the major cause regarding financial losses. These come from a not so transparent management and dishonest information that the client has access. These questions affect the world every day, most specifically the insurance sector. The possibility of changing a client's contract policy without his knowledge nor consent turns out to be a serious inconvenience towards situations where they might need to report the damages to the insurance company. Due to the situation, a system that grants integrity, confidentiality, transparency, and security for their clients is needed. Blockchain is a platform that can provide all these characteristics.

In this dissertation, a study was made to find the most fitting blockchain platform, to prevent the possibility of fraud commitment that the insurance industry is lacking. After selecting the platform, the development of a system was followed that would serve as proof of concept on how the blockchain platform is able to offer integrity, security, confidentiality, and transparency.

Keywords: Blockchain, Fraud, integrity, insurance.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Anexos

Lista de Abreviaturas

DLT - Distributed ledger technology

PBFT - Practical Byzantine fault tolerance

aBFT - Asynchronous Byzantine fault tolerance

APIs - Application Programming Interfaces

PoW - Proof of Work

PoET - Proof of Elapsed Time

ETM - Ethereum Virtual Machine

ETH - Ether

SSH - Secure Shell

1. Introdução

1.1. Funcionamento do contrato e apólice de seguro

Quando uma entidade pretende realizar um contrato de seguro, esta começa com uma simulação em que, baseado nas suas necessidades, é determinado um prémio que a entidade – tomador do seguro – terá de pagar. Se o tomador de seguro aceitar o prémio, segue-se a proposta de seguro. Esta consiste em expressar a sua vontade de celebrar o contrato de seguro, e informar o segurador do risco que pretende cobrir.

O contrato de seguro é um acordo entre o segurador e o tomador do seguro, em que o segurador assume a cobertura de determinados riscos, obrigando-se a cumprir as indemnizações ou a pagar o capital seguro em caso de ocorrência de sinistro, conforme os termos combinados entre ambos. Em contrapartida, a entidade que celebra o seguro terá de pagar ao segurador o prémio associado ao serviço, ou seja, o custo do seguro. Este acordo é, usualmente, feito através do preenchimento de um formulário já existente para esse efeito. Quando o contrato é celebrado, o segurador deve formalizá-lo através de um documento escrito, datado e assinado, que se designa apólice de seguro. Esta pode ser entregue quando o contrato é celebrado, ser enviada, posteriormente, num prazo de 14 dias, ou num prazo combinado entre ambas as partes. A duração de um contrato de seguro designa o tempo em que estão cobertos os riscos indicados no contrato de seguro, sendo decidida por ambas as partes. Salvo as partes acordarem outra duração, o contrato de seguro produz efeitos, por um ano, a partir das 00h00 horas do dia seguinte ao da sua celebração. Também, se não for acordado em contrário, o contrato de seguro feito para um ano prorroga-se, sucessivamente, no fim do contrato, por um período equivalente. Já os contratos celebrados para menos ou mais do que um ano (contratos temporários) não se prorrogam no fim do mesmo. Quando a prorrogação de um contrato é feita com sucesso, considera-se que se trata do mesmo contrato.

Um contrato de seguro pode cessar de quatro formas:

1. **Revogação:** Modo de cessar o contrato por acordo entre as partes;
2. **Caducidade:** Quando chega ao final do seu período de vigência, exceto se for automaticamente prorrogado;
3. **Denúncia:** Modo de cessar o contrato para evitar a sua prorrogação. Deve ser feita por escrito e enviada ao destinatário dentro dos prazos permitidos;

4. **Resolução:** Ocorre quando o contrato cessa por iniciativa de uma das partes. Havendo justa causa, qualquer uma das partes pode fazer cessar o contrato de seguro a qualquer momento. Também é possível desistir do contrato de seguro sem justa causa dentro de certas condições [1].

1.2. A fraude no setor financeiro

A *Association of Certified Fraud Examiners* (ACFE) (2016), caracteriza três tipos de fraude ocupacional: corrupção, apropriação indevida de ativos, e fraude nas demonstrações financeiras. A fraude nas demonstrações financeiras é a que menos ocorre, contudo, é a que tem, de longe, o maior impacto, em termos de perdas financeiras. Este tipo de fraude não é facilmente detetável, e, por norma, é descoberta anos depois do infrator a cometer.

Todos os investimentos envolvem riscos. No entanto, o problema do setor financeiro é que, enquanto os lucros são privatizados, as perdas são nacionalizadas, sendo também comum aproveitarem-se da honestidade dos clientes, para falsificação de documentos, reduzindo, assim a perda da empresa. Isto foi bem visível com a crise financeira despoletada pela falência do *Lehman Brothers*, nos EUA, em 2008, ou com os casos do BPP, BPN, BES e Banif, em Portugal [2] e com a falsificação de documentos nas seguradoras, como Prudential, no Brasil, e *State Farm*, nos EUA, ambos os casos em 2018 [3],[4].

1. Lehman Brothers:

Segunda-feira, 15 de setembro de 2008, o banco de investimento *Lehman Brothers* formalizou a maior falência na história dos EUA, que, consequentemente, provocou uma crise financeira a nível mundial. No momento da falência, a *Lehman Brothers* tinha 25 mil trabalhadores espalhados pelo mundo, 639 mil milhões de USD em ativos, e 619 mil milhões de USD em empréstimos. O negócio consistia, maioritariamente, em hipotecas de alto risco, com o nome *subprime*. Eram consideradas de alto risco pois os devedores não apresentavam capacidades favoráveis de pagamento. Para financiar os empréstimos, a *Lehman* levantava dinheiro, todos os dias, em quantias que atingiram mil milhões. Em 2007, a crise de crédito imobiliário *subprime* aconteceu, e o *Lehman Brothers* não resistiu, deixando de conseguir dinheiro para financiar os seus empréstimos. O banco perdeu cerca de 3,9 mil milhões de USD no terceiro trimestre de 2008, após ter sofrido fortes depreciações dos ativos, ao nível do seu portfólio de créditos imobiliários. Em apenas um ano, o banco caiu 90% na bolsa. Esta e outras práticas duvidosas, como uma

cultura de risco excessivo, serviços complicados com operações, e riscos espalhados por todas as estruturas corporativas, levaram à queda e falência do *Lehman Brothers*. Consequentemente, milhões de pessoas perderam os seus bens e a sua fonte de rendimento [5].

2. Banco Espírito Santo (BES):

Em Portugal, o ano de 2014 foi marcado por um dos maiores casos de fraude financeira conhecidos até hoje. No início desse mesmo ano intensificaram-se as investigações às empresas do *Grupo Espírito Santo,* sendo descoberto um prejuízo consolidado de 5.300 milhões de euros. Em maio, anunciou-se um aumento de capital, e, apesar de este ter sido efetuado com sucesso, não foi suficiente para salvar o grupo *Grupo Espírito Santo*. Poucas semanas depois, começaram a surgir as primeiras notícias sobre as irregularidades nas contas do grupo e no decorrer das investigações, o valor da dívida foi aumentando e a "almofada" de liquidez deixou de ser suficiente. No dia 3 de agosto de 2014, o Banco de Portugal anunciou a medida de resolução, e surgiu, assim, o Novo Banco, com os "capitais bons". O "banco mau" entrou, deste modo, em liquidação [2],[6].

3. Prudential:

A *Prudential* é uma empresa de serviços financeiros, com mais de 140 anos, e com presença em 40 países. Em março de 2018, foi condenada, pela justiça de São Paulo, por falsificar a assinatura do seu cliente, cliente esse que fez um seguro de vida com cobertura para doenças graves. Consequentemente este foi diagnosticado com cancro, e pediu a indemnização prevista no contrato de 100 mil reais. Foi-lhe negado o pedido, com a justificação de que, no contrato do cliente, a devida cobertura não terá sido acordada. Não reconhecendo a sua assinatura, contactou as autoridades para o caso. Foi determinada a produção da prova pericial grafotécnica, provando que a assinatura tinha sido, efetivamente, falsificada. Ou seja, a empresa alterou a apólice do cliente para uma que lhe fosse desfavorável, indo contra o seu acordo e conhecimento [3].

4. State Farm:

A *State Farm* é uma empresa de seguros e serviços financeiros, sendo a maior empresa de seguros automóvel nos EUA. Em 2018 uma dúzia de processos judiciais foram apresentados contra a empresa, por clientes vítimas de fraude pela parte da empresa e respetivos agentes. Uma cliente foi atropelada, e apelou a *State Farm*, dada a sua presumida cobertura contra o caso, mas

o pedido foi-lhe negado – afinal, não tinha cobertura. Estando convencida que tinha assinado a suposta cobertura, apelou às autoridades. Com o decorrer do caso, descobriu-se que a apólice tinha sido falsificada, tendo sido feita uma outra apólice, inferior e de menor valor, falsificando a assinatura da cliente, sem o seu consentimento ou conhecimento. Muitos outros processos judiciais revelam o mesmo problema com a empresa [4].

Todos os casos demonstram fraudes, começando por operações e tipos de gestão invisíveis à justiça. Os seus objetivos consistiam no sucesso e bem-estar da empresa e dos seus participantes, aproveitando a honestidade e meios financeiros provenientes dos seus clientes para tal. Esta situação poderia ser resolvida havendo transparência nas ações e gestão da empresa por detrás dos serviços propostos, e protegendo os dados dos seus clientes.

1.3. Segurança e transparência no blockchain

Como demonstrado na secção anterior, atualmente é extremamente fácil de abusar da confiança do cliente, cometendo fraudes, como a alteração da sua assinatura ou condições de acordos. Sendo a confiança no sector financeiro um problema, existe a necessidade de soluções que possam retirar a possibilidade de alterar documentos, sem consentimento ou conhecimento do cliente – um sistema que esteja sempre disponível para consulta, que garanta privacidade, e que possa ter acesso a não mais do que o necessário. A solução encontra-se no blockchain e na utilização de *Smart Contracts*, através do mesmo.

Blockchain é uma tecnologia de registos digitais de transações incorruptíveis, que pode ser programada para registar não só transações, como tudo o que tenha valor. O seu principal objetivo é permitir transações *peer-to-peer*, rápidas, seguras e transparentes. É uma rede confiável e descentralizada, que permite a transferência de valores digitais, como moedas e dados. Funciona à base de blocos encriptados, que servem como repositórios para as mesmas transações, e também podem armazenar porções de código que poderão ser ativados quando certas condições são atingidas, denominados de *Smart Contracts* – por exemplo, a compra de uma propriedade ou fecho de um contrato de seguro. Todos os blocos são compartilhados numa grande rede de computadores, conhecidos como nós, e são completamente públicos. Cada vez que é feita uma atualização num bloco, essa atualização é, automaticamente, descarregada e verificada em todos os computadores pertencentes à rede. Portanto, quantos mais computadores estiverem na rede, mais segura ela será. A única forma de fazer alterações fraudulentas na rede *blockchain* seria hackear mais de 50% dos computadores na rede, o que, de momento, é impossível dadas as altas proteções fornecidas através da encriptação dos mesmos. Desta forma, o *blockchain* é extremamente seguro para

transações, assim como para o seu destinatário. Isto permite eliminar uma terceira entidade para verificação das transações, diminuir os seus custos, e a sua total descentralização. O facto de todas as transações serem públicas fornece total transparência aos utilizadores que, em conjunto com o anonimato fornecido pela encriptação dos dados nos blocos, e a impossibilidade de os hackear, resulta num excelente serviço para o sector financeiro [7],[8].

Smart Contracts é uma forma de aproveitar a tecnologia de *blockchain*. *Smart Contract* é, resumidamente, um programa de computador, que funciona como um contrato, onde os termos são programados como gatilhos. O contrato entrará em execução quando uma condição do mesmo for acionada, fazendo com que todo o processo seja automatizado, sem necessidade de intervenção humana. Isto não só permite reduzir custos, por não haver necessidade de intermediários, por exemplo agentes de seguro, etc., como também oferece um serviço a toda e qualquer hora, ao dispor do cliente. No mundo das seguradoras, este serviço poderá transmitir confiança e credibilidade ao cliente relativamente às condições de oferta que este poderá usufruir, e acesso aos seus dados de cliente a qualquer momento [9].

1.4. Objetivos, resultados e estrutura da dissertação

O objetivo desta dissertação é assegurar verificar que o blockchain é capaz de ir de encontro às necessidades do setor financeiro, neste caso indústria de seguros. Fornecer confidencialidade, privacidade e transparência. Como tal no capítulo 2 é feito um estudo sobre três plataformas de *blockchain*: *Ethereum*, *HyperLedger Fabric* e *R3 Corda*. Com base nesse estudo será selecionada a plataforma mais adequada para as necessidades mencionadas anteriormente. No capítulo 3 será feito um estudo mais aprofundado da plataforma escolhida e no capítulo 4 será feito o desenvolvimento de uma pequena prova de conceito cujo objetivo será demostrar que é possível obter segurança, transparência, privacidade e confidencialidade a partir do blockchain. Para finalizar, no capítulo 5 serão as conclusões baseadas no desenvolvimento feito no capítulo 4 e o que poderia ser melhorado e, também, como é que o trabalho poderá ser desenvolvido para tarefas mais complexas.

2. Estado de Arte

2.1. Tipos de plataformas de blockchain

Há vários tipos de plataformas de *blockchain*. Apesar de todas elas partilharem o mesmo nível de segurança básico, oferecido pela tecnologia de confiança, é possível desenvolver proteções, funcionalidades e serviços adicionais, tornando-os diferentes entre si. Neste capítulo, serão analisadas as plataformas *R3 Corda*, *Hyperledger Fabric* e *Ethereum*, por serem plataformas populares no desenvolvimento de aplicações com *Smart Contracts* em *blockchain*.

Para o ramo das seguradoras, é necessário que a plataforma escolhida para o objetivo desta tese ofereça confidencialidade, segurança, acesso permanente, meios de autenticação e identificação. Como as três plataformas estão baseadas no *blockchain*, estas já oferecem meios de segurança e acesso permanente, e distinguem-se no nível de privacidade e confidencialidade.

Antes de se proceder à análise de cada plataforma, há termos que têm de ser descritos pela sua essencialidade na diferenciação entre plataformas de *blockchain*, e na escolha destas três plataformas em particular [10].

Os termos anteriormente mencionados baseiam-se em diferentes características, dessas, as mais importantes são:

- **Algoritmo de consenso:** O setor financeiro, atualmente, é composto por uma rede centralizada, ou seja, só há uma entidade que mantém a cópia da base de dados. Isto fornece-lhe um grande controlo sobre os dados permitidos e os não permitidos, assim como a decisão relativamente ao que os seus utilizadores podem ver. No *Distributed ledger technology (*DLT), as múltiplas entidades contêm uma cópia da base de dados, e têm permissões para contribuir informação. Desta forma, todos constroem uma rede de nós, gerando um problema, visto que se alguma alteração ocorre, estas têm de se propagar para todos os outros nós na rede. Ou seja, têm de chegar a um estado comum. O resultado deste processo chama-se de consenso entre nós. Este é um grande desafio nas diferentes redes de blockchain pois um algoritmo de consenso que seja resiliente a falhas nos nós, mensagens atrasadas e/ou fora de ordem e também mensagens corruptas, etc., pode vir com consequências ou limitações de um de serviço na rede. As características mais importantes no algoritmo de consenso é garantir que cada nó tem capacidade de decidir sobre um determinado valor, segurança de forma a garantir que nós diferentes não irão concordar com decisões diferentes e *fault-tolerance* que representa a capacidade de

recuperação da rede numa situação em que um nó malicioso participou no consenso. Em *DLTs* com maior frequência ocorrem dois tipos de falhas: falhas de paragem e falhas de *Byzantine*. As falhas de paragem ocorrem quando há uma interrupção do funcionamento do software ou hardware o que impossibilita os nós de continuar a participar no consenso. As falhas de Byzantine ocorrem em menos quantidade e ocorrem quando uma entidade está a tentar alterar a identidade de um nó ou quando há problemas de software relacionados com o mesmo. [11],[12],[44].

- **Permissividade da plataforma (tipo de registo):** Existem duas formas diferentes que uma plataforma pode utilizar, para que o utilizador possa fazer parte da rede. Registos com permissão e registos sem permissão. Os registos sem permissão, como o *Ethereum*, implicam que qualquer utilizador pode tornar-se membro da rede e ter todos os direitos para a utilizar. No entanto, num registo com permissão, como no *Hyperledeger Fabric* e *R3 Corda*, isto não acontece, visto que os utilizadores têm de cumprir um determinado número de características antes de terem acesso à rede. Em alguns casos, os acessos são restritos, mesmo a quem tem autorização, assim como são submetidos a uma seleção adicional para distinguir os utilizadores que terão acesso a mais dados do que outros. Isto é bastante útil quando queremos utilizar os *Smart Contracts*, onde só os utilizadores que forem autorizados têm acesso à rede [13].

- **Encriptação funcional por *hash*:** Os sistemas de *blockchain* funcionam em larga escala à base de encriptação por *hash*, que consiste na transformação da informação original. A encriptação do *hash* é um algoritmo que mapeia dados de tamanho arbitrário com o objetivo de os tornar impossíveis de reverter. Este tem três graus de dificuldade, sendo o primeiro denominado de *preimage resistance* em que é computacionalmente difícil de reverter a função *hash.* De seguida, uma segunda camada de *preimage resistance* é aplicada em que se torna computacionalmente difícil de encontrar uma informação diferente, mas com um *hash* semelhante. Por fim é aplicada uma terceira camada de *collision resistance* para tornar computacionalmente difícil encontrar dados diferentes de qualquer tamanho que possam obter um *hash* igual ou semelhante.

  Um exemplo prático é o uso da crypto moeda *Bitcoin*. No sistema de pagamento do *Bitcoin* um sistema de encriptação de pares de chaves públicas e privadas é aplicado. A chave privada é escolhida através de uma função de curva elíptica. De seguida um processo de encriptação por *hash* é aplicado para criar a chave publica e mais um processo de *hash* é utilizado em cima desse para criar o endereço de Bitcoin. Desta forma temos

imensos meios de mascarar a identidade e dados de cada passo no processo de uma transação [10],[11],[44].

- ***Smart Contracts:*** Estes trazem o valor empresarial ao *blockchain* além das transferências monetárias. Consiste na combinação de protocolos e interfaces de utilizador para validar e assegurar relações entre redes de computadores. Na prática são porções de código modulares implementados no *blockchain* que asseguram premissas unilaterais para criar uma computação determinada. Pedaços de código são mantidos em certos endereços no *blockchain*, sendo o endereço determinado no momento de validação do contrato pelo *blockchain*. Quando ocorre um evento descrito no contrato, uma transação é executada. Esta transação pode conter tanto dados informativos, como documentos, etc., como também transação de bens valiosos. É de notar que uma vez que um *Smart Contract* é registado e implementado no blockchain, torna-se imutável e inconvertível. Para um *Smart Contract* ser implementado e registado numa rede *DLT*, tem que ser aprovado por todos os nós na rede. É possível tornar os contratos mais complexos com várias sequencias de subcontratos, repositórios de informação, etc. Há várias plataformas de Blockchain e cada uma tem o seu método de implementação e registo de *Smart Contracts*, sendo o mais popular o da rede *Ethereum* [10],[11],[23],[44].

2.2. *R3 Corda*

Desde 2016, o *R3 Corda* consiste num registo de código aberto, e num conjunto de padrões que se subdividem na gestão dos processos de rede e nos seus parâmetros. Coletivamente definem uma rede *Corda*. Deste modo, possibilitam qualquer organização ou utilizador individual a transitar desta rede aberta para qualquer outra. Unicamente, esta arquitetura foi construída para modelar e automatizar as transações no mundo real, de forma a forçar a legalização dos mesmos. É igualmente conseguido numa rede aberta, onde há múltiplas aplicações interligadas, e a funcionar simultaneamente. Enquanto força o lado legal, também consegue manter a identidade dos processos, a fiabilidade das transações, a privacidade de dados, e a gestão aberta. Isto permite-lhe lidar com as transações mais complexas, resultando num alto interesse de vários ramos financeiros, como os cuidados de saúde, seguradoras, bancos e governos [14].

A visão do *R3 Corda* é um mundo em que as instituições/organizações/empresas conseguem manusear contratos, legalmente aprovados, e transferir valores, sem restrições tecnológicas, ou perda de privacidade. Em contraste às plataformas de *blockchain* sem permissões, o Corda

tenciona manusear transações do mundo real, entre partes identificáveis, enquanto mantém a privacidade, e reforça a legalidade. Em contraste às plataformas de blockchain com permissões, o *Corda* tenciona autorizar múltiplos grupos participantes, e as suas aplicações, a coexistir e operar na mesma rede aberta. O modelo de gestão está desenhado para refletir os interesses da diversa base de utilizadores na plataforma, fazendo com que o *Corda* suporte uma variedade de funções, tanto no ramo de redes com permissões, como nas redes sem permissões, tornando-o extremamente flexível.

Para facilitar o estabelecimento de obrigações que surgem dos contratos manuseados na plataforma, o *Corda* permite a emissão, transferência e retorno de valores, onde a regulação o permita. Em adição, a arquitetura permite a emissão de ativos nativos e/ou qualquer moeda quer no formato original ou mascaradas sob *tokens (*moedas/fichas criadas por um utilizador na rede para representar uma moeda) na plataforma, o que incentiva a adoção, participação e pagamento de serviços, abrangendo uma variedade de aplicações particulares a uma instituição/organização/empresa que operem na rede *Corda*.

Os princípios da visão final do *R3 Corda* que pretendem alcançar, para as empresas/organizações/instituições, é a inclusão, em que as partes conseguem descobrir-se umas às outras, livremente, e conseguirem transações, diretamente, numa rede aberta. Assegurar igualmente a identidade dos participantes, pois haverá segurança sobre cada participante na rede. Privacidade, pois, as únicas partes que terão acesso à informação de uma transação serão as próprias partes que participaram na transação, e as partes que estas autorizarem. Imutabilidade, pois os factos gravados nos registos serão finais, inclusive erros, que serão processados com transações subsequentes. Transparência, onde todo o sistema é aberto, no sentido em que a fonte de código, a participação, o desenvolvimento, a gestão, e os standards que asseguram os balanços necessários para a diversa base de utilizadores será completamente lúcido. As visões do *Corda*, para a arquitetura da plataforma, incluem escalabilidade, para permitir mil milhões de transações diárias entre indústrias. Longevidade, o que tornará possível que as várias versões diferentes do *Corda* existam, simultaneamente, na rede, e consigam trabalhar e funcionar a 100 por cento. Estabilidade, dado que a evolução da tecnologia irá ser cuidadosa, com decisões manuseadas num ambiente de gestão aberta. Interoperabilidade, que fornece à plataforma a possibilidade de ter múltiplas aplicações, coexistindo e operando na mesma rede, com standards predefinidos para as interfaces dos seus contratos, que maximizam a interoperabilidade para a grande variedade de fornecedores de serviços.

O *R3 Corda* é diferente do *Ethereum* e do *Hyperledger Fabric* (analisados mais abaixo), porque a gestão da rede é feita através de *Smart Contracts* que têm a capacidade de fazer verificações de validade, até ao nível de assinaturas digitais, e definir o que os utilizadores podem, ou não, fazer ou ver, dentro da rede. Não há limite para o número de *Smart Contracts* a trabalhar em simultâneo, pelo que permite à plataforma ter um grande grau de escalabilidade. Este é o forte fator no mundo das seguradoras [15].

O algoritmo de consenso, no *R3 Corda*, consiste no algoritmo *Asynchronous Byzantine Fault Tolerance* (aBFT), onde a validade e a unicidade de uma transação são sujeitos de consenso. Apenas quando as duas condições se verificarem é que uma transação é efetuada. A validade é assegurada por correr um *Smart Contract*, relacionado com uma transação específica. Este verifica se a transação contém as assinaturas necessárias, ou seja, de todos os participantes diretos na transação, e verifica a validade da transação, relacionada com a assinatura. A unicidade da transação é fornecida e validada por nós, designados de "nós notários" (*notary nodes*). Quando um nó pede a assinatura do notário, este assegura-se que a mesma proposta é válida, quanto à sua unicidade de consenso, e, se for, assina e envia-a de volta para o nó. Se a proposta não passar a verificação, vai ser rejeitada e assinalada como tentativa de duplo gasto – quando um utilizador consegue duplicar a sua moeda, dando-lhe uso múltiplas vezes, sem a gastar-. O nó, ao receber a informação sobre a possibilidade de a proposta ser um duplo gasto, descarta-a por completo. Desta forma há um reforço sobre cada transação se referir a um único consumidor, não permitindo mais transações desse tipo. Isto resolve o problema do duplo gasto e fornece ao *R3 Corda* uma excelente segurança, confidencialidade e privacidade [16],[17].

A agilidade algorítmica do *R3 Corda* é uma área de intensa investigação, com novos algoritmos sendo constantemente desenvolvidos, melhorando o seu estado de arte. Ao contrário de outras plataformas de *blockchain*, o *Corda* não se agarra a um algoritmo específico de consenso. Isto não só é favorável para o apoio de melhoramentos no aparecimento de novos algoritmos, como também reflete, conforme a situação em que nos encontrarmos, a possibilidade de utilizar o algoritmo mais apropriado. Por exemplo, numa rede em que os nós são de confiança, podemos utilizar um algoritmo de segurança reduzida, e, em troca, mais velocidade e menos latência são fornecidas. Em contrapartida, se estivermos numa rede, com parceiros desconhecidos, faz sentido utilizar o algoritmo aBFT, em que a velocidade é menor e a latência é maior, mas é obtida uma segurança muito mais reforçada [18].

A fonte de inspiração para a criação do *Corda* consiste na criação de um sistema de transação, em que ambas as partes confiam uma na outra para realizar a transação, mas não o

suficiente para que a outra parte aceda a todos os registos e dados. Isto torna o *Corda* num excelente candidato para o objetivo desta tese, visto abranger diretamente todos os requisitos que procuramos. Não só os abrange, como também tenciona ser o melhor no ramo, com constantes atualizações e desenvolvimentos, melhorando a qualidade de vida dos utilizadores [15][17].

2.3. *Hyperledger Fabric*

Em 2015, quando muitas empresas interessadas no *blockchain* perceberam que conseguiriam alcançar mais se trabalhassem em conjunto, ao invés de trabalhar individualmente. Juntas criaram uma infraestrutura, denominada de *Hyperledger*, de código aberto, na tecnologia *blockchain*, que qualquer pessoa poderá utilizar. O *Hyperledger* foi colocado sob a gestão da *Linux Foundation*, e tem crescido rapidamente nos últimos anos. O facto de ser código aberto tem sido uma ajuda considerável para essa causa [19].

A filosofia por trás do *Hyperledger* consiste em ser flexível ao nível da otimização na transferência de valores, conforme as diferentes situações. Por exemplo, numa situação em que há transferência de dados entre duas partes, com alto nível de confiança, deveriam ser capazes de reduzir o nível de segurança e receber mais velocidade de transferência. Já quando a confiança é pequena, o nível de segurança pode ser aumentado, e tolerando menos velocidade. Sendo assim, o *Hyperledger* reconhece que cada situação tem o seu ponto de otimização. Para apelar a esta diversidade, todos os projetos do *Hyperledger* seguem a mesma filosofia de design. Todos os projetos *Hyperledger* devem ter várias características: modulares; altamente seguros; interoperáveis; agnósticos a criptomoedas; completos com *Application Programming Interfaces* (APIs) [12].

1. **Modular:** *Hyperledger* desenvolve uma extensa estrutura modular, com blocos de registo básico, que podem ser reutilizados. Isto permite aos desenvolvedores a experiência com diferentes componentes, à medida que evoluem, e a construção de aplicações específicas, para uma variedade de indústrias que apelem às suas necessidades. O *Hyperledger Architecture Working Group* define módulos funcionais para problemas, tais como a comunicação, consenso na rede, criptografia, identidade, armazenamento nos registos e *Smart Contracts*.
2. **Altamente Seguro:** É um assunto importantíssimo para o *Hyperledger*, especialmente porque transações de valores altíssimos ou de informação com alto grau de sensibilidade podem acontecer. Os projetos são desenvolvidos com a segurança em primeiro

lugar, seguindo as melhores práticas de uso, especificadas pela *Linux Foundation's Core Infrastructure Initiative*. Como tal, todos os algoritmos, protocolos e métodos de encriptação do *Hyperledger* são regularmente revistos e auditados por especialistas na área, também como, por ser de código aberto, é revisto pela grande comunidade por trás do *Hyperledger*.

3. **Interoperável:** No futuro, muitas redes diferentes de blockchain terão de conseguir comunicar e trocar dados, para criar redes mais complexas e com maior potencial. O *Hyperledger* acredita que a maior parte dos *Smart Contracts* e aplicações deveriam ter a capacidade de serem integrados em todas as redes de *blockchain*. Assim, o alto grau de interoperabilidade aumentará a adoção do *blockchain* e as suas respetivas tecnologias.
4. **Agnóstico a criptomoedas:** O *Hyperledger* é agnóstico a todas as criptomoedas, moedas alternativas ou *tokens*, dado que o objetivo é assegurar a transferência de valores, com qualquer moeda que as partes escolherem utilizar. Pretende criar software na tecnologia *blockchain* para empresas, e não assegurar a existência ou apoio a uma cripto moeda. No entanto, o design do *Hyperledger* não impede a criação de um *token* utilizado para manusear objetos digitais, podendo representar um tipo de moeda. Mas é totalmente opcional, pois não é um requerimento para a rede conseguir operar.
5. **Completo com APIs:** Todos os projetos do *Hyperledger* fornecem APIs excelentes e fáceis de utilizar, que apoiam a interoperabilidade com outros sistemas. Um conjunto bem definido de APIs permite a interação entre clientes e aplicações externas, com rapidez e facilidade, contando com o apoio do núcleo da infraestrutura do *Hyperledger*. Assim, os APIs fornecem um rico crescimento dum ecossistema de desenvolvedores e ajudam o *blockchain* e as suas tecnologias a proliferar num grande número de indústrias e casos de uso diferentes [12].

De momento o *Hyperledger* incuba e promove várias estruturas de *blockchain*, tais como as seguintes: Hyperledger Burrow; Hyperledger Fabric; *Hyperledger Indy; Hyperledger Iroha; Hyperledger Sawtooth* [20].

- ***Hyperledger Burrow*:** Tecnologia *blockchain* modular, baseada num interpretador com permissão, operado através de um *Smart Contract*. Foi desenvolvido, em parte, com as especificações da *Ethereum Virtual Machine* (EVM);

- ***Hyperledger Indy*:** Um registo distribuído que fornece ferramentas, bibliotecas e componentes reutilizáveis, desenvolvidos com de propósito para as identidades descentralizadas;
- ***Hyperledger Iroha*:** Uma estrutura *blockchain* desenhada para ser simples e fácil de integrar nas infraestruturas e projetos empresariais;
- ***Hyperledger Sawtooth*:** Uma estrutura modular para a construção, envio e funcionamento de registos distribuídos. *Sawtooth* fornece um novo tipo de consenso, *Proof of Elapsed Time* (PoET), que gasta muitos menos recursos que o *Proof of Work* (PoW);
- ***Hyperledger Fabric*:** Uma plataforma especificamente designada para o desenvolvimento de soluções, utilizando registos distribuídos com uma arquitetura modular, que fornecem um alto grau de confidencialidade, flexibilidade, resiliência e escalabilidade. Isto permite que as soluções desenvolvidas através do *Fabric*, possam ser rapidamente adotadas a qualquer indústria [12],[20].

O algoritmo de consenso utilizado no *Fabric* é o *Practical Byzantine Fault Tolerance* (PBFT), que fornece aos nós a capacidade de abranger todo o processo de uma transação, e desempenhar funções diferentes, conforme a sua identificação, para alcançarem o consenso. Os nós podem ser identificados como clientes, solicitadores, ou simplesmente nós. Os clientes funcionam como utilizador final, pelo que invocam transações e interagem com os nós e solicitadores. Os solicitadores fornecem um canal de comunicação para os clientes e os nós, tornando possível o envio de mensagens que contenham transações. Por fim, os nós são os que mantêm o registo de dados, e recebem mensagens dos solicitadores com novas transações. Este tipo de consenso possibilita um alto grau de escalabilidade, segurança, privacidade e confidencialidade, mas, no entanto, não permite tanta flexibilidade nestes atributos, quando comparado ao *R3 Corda* [21].

Concluindo, uma rede *Fabric* lida com os nós dos pares que executam o *Chaincode*, assim como com o acesso aos dados de registo que aprovam as transações e as interfaces com as aplicações. Também consegue trabalhar e integrar linguagens de programação, tais como o *Go*, *Java* e *Javascript*, tornando-o mais flexível quando comparado com outras plataformas de *blockchain* [12].

2.4. *Ethereum*

O *Ethereum* foi criado por Vitalik Buterin (1994), em 2014, e oficialmente lançado em 2015. Ao contrário do *Hyperledger* e do *R3 Corda*, o *Ethereum* tem um *token*/criptomoeda próprio, denominada de *Ether* (ETH). O ETH é utilizado para fazer transações de valor dentro da

plataforma. O *Ethereum* não só tem uma cripto moeda, como também é uma plataforma virtual com base no *blockchain* [22].

Sendo uma plataforma virtual, o *Ethereum* tem uma máquina virtual própria [*Ethereum Virtual Machine* (EVM)], que fornece o serviço de *Smart Contracts* programáveis, em *Solidity*, onde são necessários ETHs para a sua utilização. Isto significa que, se quisermos implementar *Smart Contracts* com venda e compra de valores, estes terão de ser feitos, utilizando a moeda ETH. Também é necessário gastar ETH para os colocar a correr. Quanto maior for o código do contrato a implementar, maior será a taxa a pagar para este ser executável. Isto incentiva os desenvolvedores a implementarem aplicações pequenas [23]. O valor dos ETH depende do mercado que, de momento, tem o valor de 200 euros por 1 ETH [24]. Em contrapartida, as plataformas *R3 Corda* e *Hyperledger* não necessitam de recursos financeiros para executar *Smart Contracts*, para além de permitirem implementar a moeda desejada para as transações entre utilizadores.

O algoritmo de consenso do *Ethereum* é baseado na prova de trabalho (*Proof-of-Work*), que consiste em ter todos os nós em concordância no mesmo segmento de registos, e todos os nós têm acesso obrigatório a todas as transações feitas e gravadas. O algoritmo é que impõe a dificuldade e as regras a que os mineiros se sujeitam. Os mineiros são quem fazem o "trabalho". Este consiste em adicionar blocos válidos à sequência de blocos no *blockchain*. Isto é importante porque o comprimento da corrente de blocos ajuda a rede a perceber qual é a corrente válida e entender o seu estado. Quanto mais 'trabalho' for realizado, maior será a corrente, o número de blocos, etc. Os mineiros passam por uma corrida intensa de tentativa e falha para desencriptar o próximo bloco a ser criado na rede. Para poder fazer isso têm que passar um conjunto de dados, que só conseguem obter por fazer download de todas as transações registadas e correr todos os blocos anteriormente criados na rede através de uma equação matemática. Quanto maior for a rede, mais blocos terá e consequentemente, a nível de energia e hardware, mais dispendioso se torna pôr um novo bloco na rede. Uma vez que o novo bloco foi criado, ele será adicionado à corrente mais comprida, ou seja, a corrente com mais blocos no momento, esta escolha é baseada no facto de que uma corrente maior é também a que teve um maior processo computacional feito e com isso será mais segura e credível. Para poder criar blocos maliciosos, no entanto válidos, um atacante precisaria de 51% do poder de processamento da rede para estar a frente de todos, o que lhe permitiria ser o primeiro a desencriptar novos blocos e adicioná-los à rede. De momento, dado o tamanho da rede *Ethereum*, isso é impossível. Isto permite uma excelente proteção, mas pouca privacidade, dado que todas as transações serão visíveis para todos os membros da rede, assim como pouca velocidade, visto que todos os nós têm de aprovar todas as transações que, consequentemente, também consome imensos recursos energéticos e tempo [11],[23].

De momento, a maior crítica ao *Ethereum* é o pormenor dos sérios problemas de escalabilidade que ainda reporta, consequência do *Proof-of-Work*. A outra crítica deve-se à sua pouca flexibilidade – o nível de proteção não pode ser ajustável em conformidade com a situação, ao contrário do *R3 Corda* e *Hyperledger* [11],[22],[23].

2.5. Conclusão

Resumindo as análises das plataformas numa tabela, obtivemos os seguintes resultados:

Tabela 2.1. - Tabela comparativa do *Ethereum* com o *Hyperledger Fabric* e *R3 Corda*.

	Ethereum	*Hyperledger Fabric*	*R3 Corda*
Tipo de indústria	Intersetorial	Intersetorial	Intersetorial
Algoritmo de Consenso	PoW	PBFT	aBFT
Gerência	Distribuído por todos os participantes	*Linux Foundation*	*R3*
Smart Contracts	Sim	Sim	Sim
Tipo de registo	Sem permissão	Com permissão	Com Permissão
Privacidade	Não	Sim	Sim
Linguagens de Programação	*Solidity*	*Go, Java*	*Kotlin/Java*
Moeda Própria	Sim	Não	Não

A partir da tabela, notamos que, para o objetivo da tese, o *Ethereum* não é um bom candidato. Falta-lhe privacidade, confidencialidade, flexibilidade, e é dispendioso. Já o *Hyperledger Fabric* e o *R3 Corda* são ambos excelentes candidatos. O que os distingue é o algoritmo de consenso, onde o aBFT, utilizado pelo *R3 Corda* que, com base nas características anteriormente descritas e para o objetivo da tese, é um candidato melhor. Este fornece mais flexibilidade, em comparação com o PBFT, utilizado pelo *Hyperledger Fabric*. Portanto, em suma, o futuro trabalho desta tese será desenvolvido na plataforma *R3 Corda* [16],[17].

3. Funcionamento do *Corda*

Para desenvolver aplicações e sistemas na plataforma *R3 Corda*, é necessário, antes de tudo, entender os seus conceitos chave. Sendo assim, este capítulo é subdividido em duas partes, cujos objetivos são a descrição e explicação dos conceitos chave em questão. A primeira parte é dedicada à compreensão da rede *Corda*, enquanto a segunda parte é dedicada aos conceitos por trás do desenvolvimento de aplicações na tecnologia em questão. Estes dados são necessários pois no capítulo a seguir irei prosseguir com o desenvolvimento de uma prova de conceito.

O sucesso no desenvolvimento de aplicações no *R3 Corda* segue-se pela compreensão dos seguintes conceitos [1],[2]:

Parte 1:

- **A rede *Corda:*** O ecossistema em que o *R3 Corda* existe;
- **Os registos:** Como os factos são colocados e partilhados entre os nós pertencentes à rede.

Parte 2:

- **Estados:** Os estados apresentam factos partilhados nos registos;
- **Transações:** As transações atualizam os estados nos registos;
- **Contratos:** Os contractos determinam a validade de uma proposta de transação;
- **Fluxos:** Os fluxos descrevem as interações que devem ocorrer entre os nós, para alcançar o consenso, e como a informação é partilhada na rede;
- **Consenso:** Como é que os nós na rede conseguem chegar ao consenso sobre os factos partilhados;
- **Notários:** Componente utilizado para assegurar a unicidade de consenso, prevenindo o duplo gasto (explicado mais à frente).

3.1. Funcionamento do Corda - Parte 1

3.1.1. A rede *Corda*

A rede *Corda* é uma rede *peer-to-peer* [26], composta por nós. Cada nó representa uma entidade legal, onde cada um corre o software *Corda* (um ou mais *Smart-Contracts*, também denominados de *CorDapps*). Todas as comunicações entre nós são ponto-a-ponto, encriptadas, utilizando a segurança da camada-de-transporte [27]. Isto significa que os dados são partilhados apenas quando necessários, para o conhecimento do(s) nó(s). Todos os nós têm a capacidade de comunicar entre si, mas só apenas se for necessário. Isto permite a segurança do *Corda*, e ter também um melhor controlo sobre o congestionamento da rede. Em adição, se um nó estiver *offline* enquanto quisermos comunicar com o mesmo, a mensagem fica em fila de espera, na camada de rede, e aguarda pelo utilizador voltar a ficar online, para aí ser enviada. Isto remove a necessidade de persistência das conexões entre nós, se os mesmos assim o optarem [28].

A identidade de cada nó é única, e reconhecida por outros nós na rede. A sua identidade é utilizada para o representar nas transações, como por exemplo, durante a participação do nó na compra de ativos.

Para apresentar e encontrar os nós, existe um mapa de rede que associa cada nó a um endereço de internet [*Internet Protocol (IP) Address]*. O mapa de rede fornece serviços, tal como um livro de contactos telefónicos, em que os contactos são os nós, e mostram-nos quem eles são, e que serviços têm disponíveis no momento (Figura. 3.1).

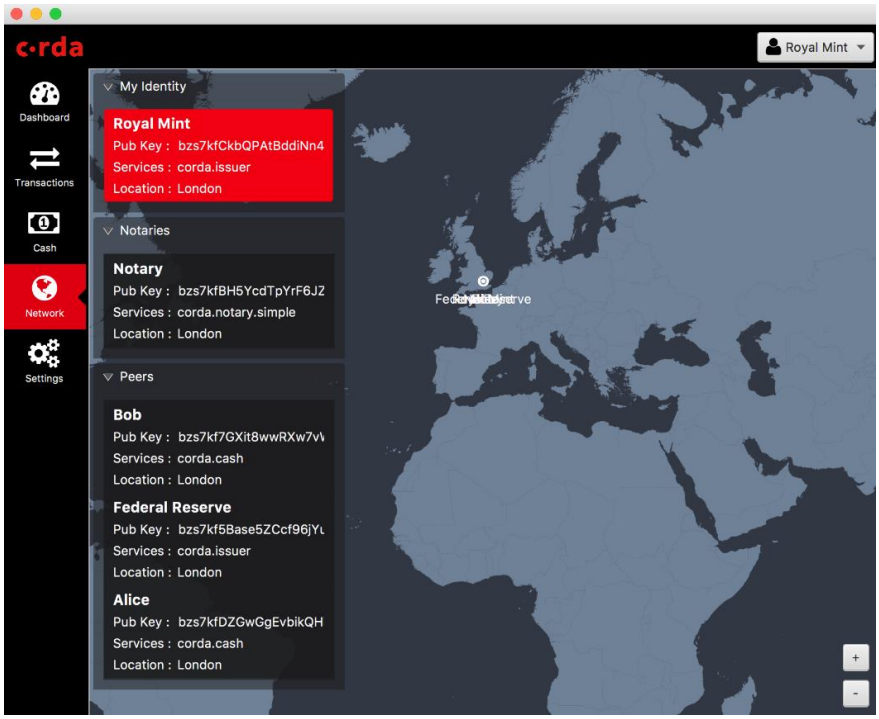

Figura 3.1. – Mapa de Rede *Corda*. Retirado de [25].

Sendo o *Corda* muito flexível na sua camada de segurança, os nós podem optar por criar identidades confidenciais para transações individuais. Assim, como a verdadeira identidade dos nós só é partilhada quando necessária, por exemplo, quando o nó notário faz a verificação da legalidade da transação, podemos estar seguros relativamente aos casos em que as transações são feitas, sem meios de encriptação. Se um atacante capturar a informação, este não conseguirá identificar os nós participantes na transação, visto estes serem confidenciais.

Por fim, para poder fazer parte da rede *Corda*, temos de ter em conta que o *Corda* é diferente das redes de *blockchain* tradicionais. As redes *Corda* são semiprivadas, portanto, para termos acesso às mesmas, temos de obter um certificado de aprovação do operador de rede. Este certificado confirmará a identidade legal do nó, fornecendo-lhe uma chave pública. O operador da rede pode, também, forçar as regras que achar necessárias, relativamente à informação partilhada pelos nós, e à mínima informação que estes devem requisitar, para que um nó se possa conectar aos restantes, ou celebrar uma transação. Se o operador achar necessário, pode tornar estes requisitos como o mínimo necessário, para o fornecimento do certificado de aprovação de acesso à rede. Esta camada extra fornece ao *Corda* mais controlo e segurança às redes [1]–[3].

3.1.2. Os registos

No *Corda,* não existe um banco central de registos. Muito pelo contrário – perante a perspetiva de cada nó, todos os registos são subjetivos. Cada nó guarda o seu próprio banco de dados, consistindo apenas em factos que ele próprio conhece.

Os factos que um nó conhece são apenas aqueles em que está envolvido. Suponha-se que existem dois nós denominados de Alice e Bob. A Alice faz uma transação de 100 unidades monetárias ao Bob, e o Bob recebe a transação com sucesso. Ambos terão um facto partilhado, sendo esse a transação em questão, e só estes dois, em toda a rede, é que o saberão. Também serão só eles a armazenar os dados da dita transação. Isto acontece porque só estes dois é que foram envolvidos na transação e no seu processo. Isto resulta num design de extrema segurança e privacidade.

Para visualizar melhor a partilha de factos nos registos, podemos observar na Figura 3.2:

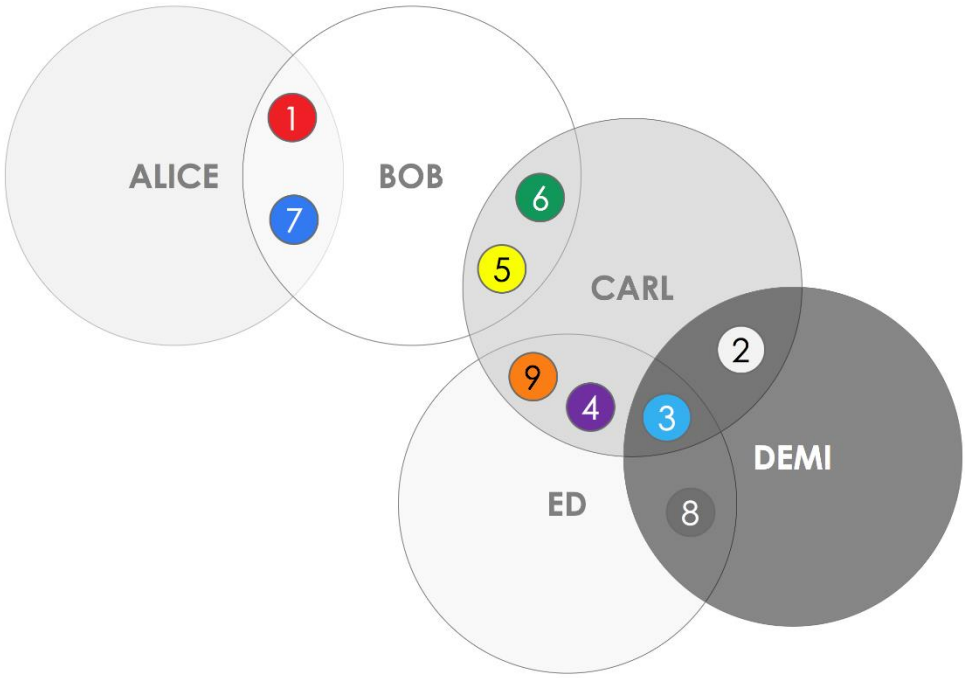

Figura 3.2. – Exemplo de um diagrama de partilha de factos no *Corda*. Retirado de [29].

Na Figura 2, temos a representação de um exemplo de uma rede com cinco nós: ALICE, BOB, CARL, DEMI e ED. Todos eles veem o banco de registos de forma diferente, porque cada um deles conhece certos factos que outros não conhecem. Verifica-se que ALICE e BOB partilham do facto 1 e sete 7, porém, BOB partilha igualmente o facto 5 e 6 com CARL. Portanto, ALICE só sabe dos factos 1 e 7, enquanto BOB conhece os factos 1, 5, 6 e 7. CARL é o que

conhece mais factos, pois é o que participou mais, partilhando os factos 5 e 6 com BOB, os factos 4 e 9 com ED, o facto 2 com DEMI, e o facto 3 com ED e DEMI. Por fim, temos o facto 8, partilhado entre ED e DEMI. Este é um puro exemplo de como a partilha de factos ocorre numa rede *Corda*.

Se um nó desejar, tem a possibilidade de tornar um facto público, dando, assim, a conhecer a existência do mesmo a todos os nós da rede. Para isso, o facto teria de ser conhecido apenas pelo nó inicial, sem antes não ter sido partilhado, se as condições desse mesmo facto o implicarem [15],[29].

Resumindo, o armazenamento dos registos no *Corda* pode ser visualizado como tabelas de dados, com identificadores para os nós com quem esses factos são partilhados. É seguro, não congestiona a rede, e proporciona privacidade [30].

3.2. Funcionamento do Corda - Parte 2

3.2.1. Estados

Os estados representam os factos guardados nos registos de cada nó. Estes podem conter informação de qualquer tipo, tal como informação de identidade, ativos, entre outros. Para atualizar um estado, é necessário marcá-lo como histórico – um facto, no passado, que já foi atualizado – e criar um estado atualizado. O estado marcado como histórico será substituído, no registo, pelo estado atualizado. Este estado marcado como histórico, uma vez substituído, será, também, movido para a cápsula de armazenamento dos nós, para manter a corrente de estados, e, consequentemente, o histórico de todas as transações nesse nó. Desta forma, passamos a ter dois estados: um oficial, sendo este o registo atual, por trás uma corrente de estados, à qual chamamos de histórico, como demonstrado na Figura 3.3:

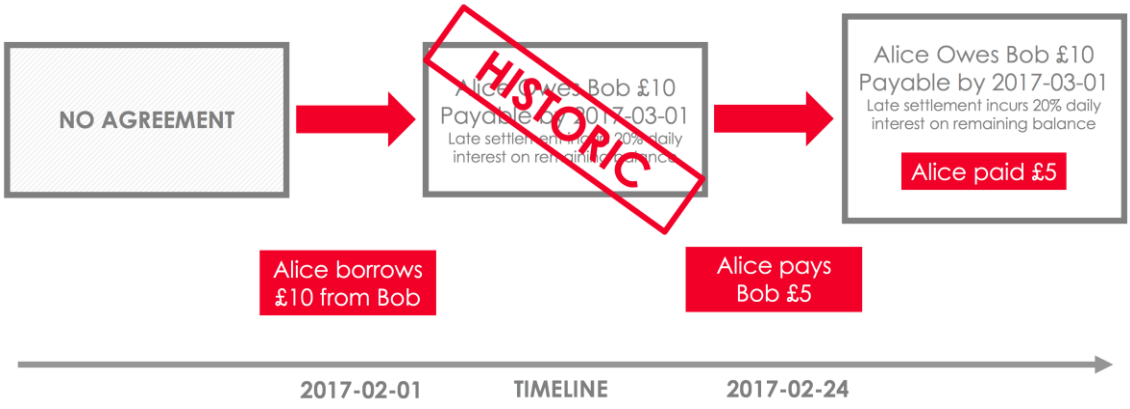

Figura 3.3. – Corrente de estados no registo de um nó. Retirado de [31].

O que observamos na Figura 3.3 é que, dada a imutabilidade dos estados, não podem ser diretamente modificados para fazer qualquer mudança. Por isso, quando um estado necessita de ser alterado, ele é copiado, e, consequentemente, atualizado, criando-se um novo. Assim, o antigo é marcado como histórico, e o novo passa a ser o novo estado real. Deste modo, criamos uma sequência de estados, que nos permite ver a evolução dos factos, ao longo do tempo. Neste caso, Alice pediu um empréstimo de 10 libras a Bob. Como Alice lhe devolveu 5 libras, surgiu a necessidade de atualizar o estado. Então, copia-se o estado, atualiza-se a informação, tornando-se, assim, como novo, enquanto que o anterior é marcado como histórico.

Como o *Corda* é extremamente flexível, os nós têm a capacidade de guardar várias sequências de estados. Uma vez que cada nó pode disponibilizar vários serviços diferentes, há necessidade de manter a sequência da evolução dos factos de cada serviço. Então, desta maneira, os estados são capazes de fazer várias sequências, sem qualquer impedimento, como demonstrado na Figura 3.4:

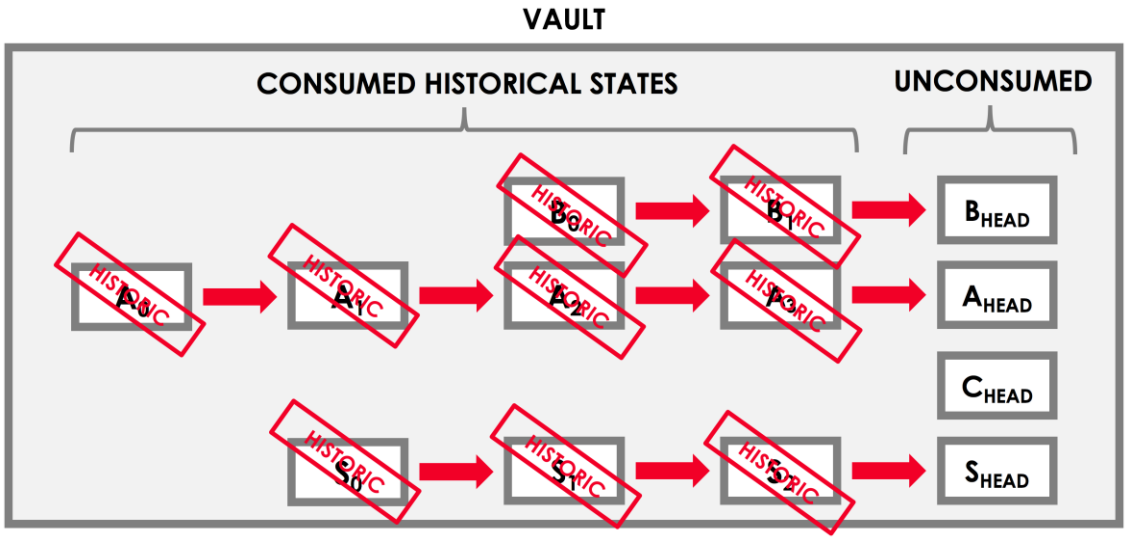

Figura 3.4. – Exemplo de várias sequências de estados que um nó pode guardar. Retirado de [31].

Na Figura 3.4, observamos que um determinado nó tem várias sequências de estados, podendo assumir que, no mínimo, possui quatro serviços diferentes. O estado atualizado também pode ser chamado de *output*, ou *head*, enquanto que os estados históricos podem ser chamados de *input* [31].

3.2.2. Transações

No *Corda*, chama-se de transações às propostas feitas aos estados, para se atualizarem. Quando ocorre uma proposta, há um ou vários campos diferentes que se quer alterar/atualizar. Para a proposta ser aceite, é necessário que todos os seus participantes a assinem, o que implica a verificação prévia de cada um, consequentemente forçando a integridade da informação. A proposta tem de ser única, não havendo gasto de recursos noutros campos da rede, ou em qualquer lado, permitindo a unicidade dos gastos, e evitando o duplo gasto, assim como outros problemas. Se, pelo menos, uma condição falhar, a proposta será recusada. Quando tudo corre bem, a proposta é aceite, passando ao estado de transação – a proposta é registada no novo estado atualizado.

Semelhante aos estados, a transação consiste no link transacional entre o estado histórico e o novo. Na Figura 3.5**,** observamos como temos uma transação simples, com duas propostas, em que o $CASH_0$ e $BOND_1$ são histórico, enquanto $CASH_1$ e $BOND_2$ são os novos estados atualizados. É de notar que no *Corda*, uma transação não tem limite para o número de *inputs* ou *outputs* que pode ter. Também podem conter qualquer tipo de informação, como dinheiro, ativos, etc. [31], [32].

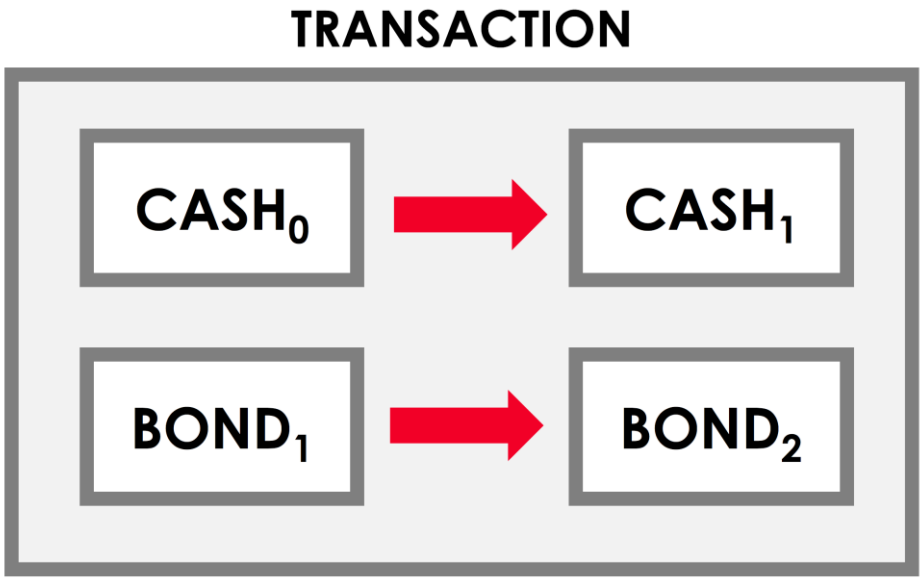

Figura 3.5. – Exemplo de dois estados de transação. Retirado de [32].

Em transações, mais complexas há necessidade de criar sequências de estados de transação, também denominados como *transaction chains*, como demonstrado na Figura 3.6:

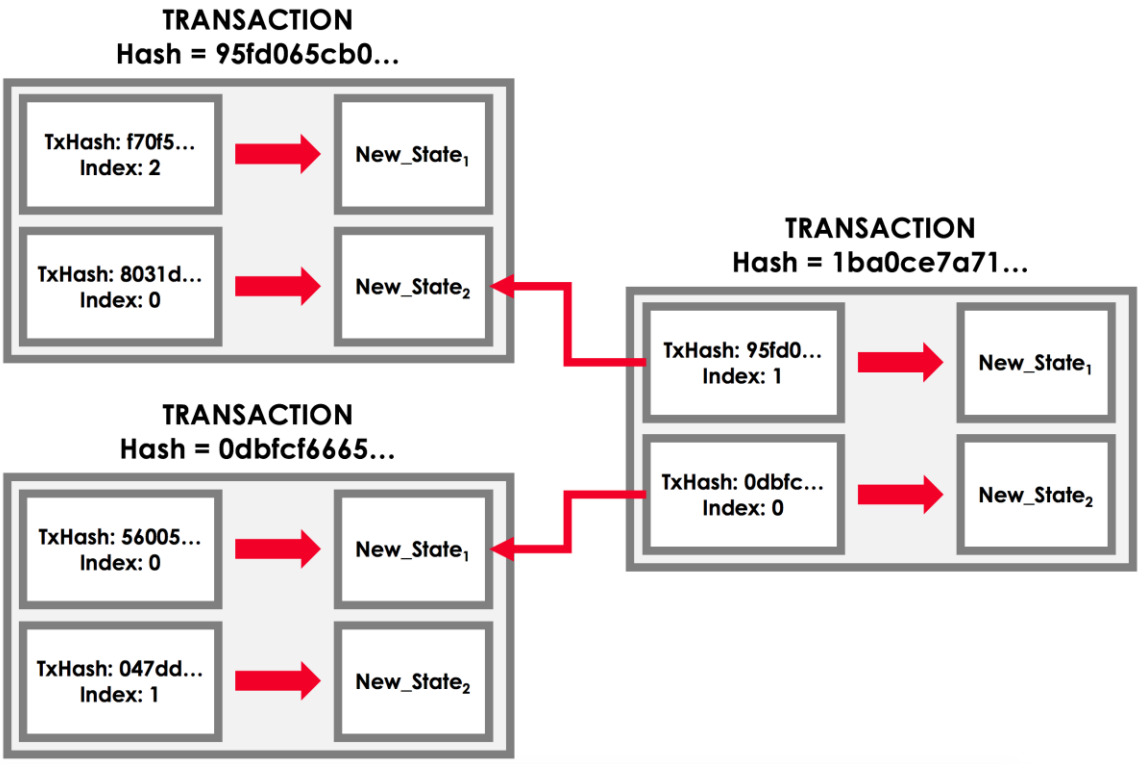

Figura 3.6. – Exemplo de *transaction chain* num nó. Retirado de [32].

Quando fazemos uma proposta que dá continuidade a outra sequência de transações, esta irá necessitar de ter como *input* a referência para o *output* da transação anterior, e utilizar o *hash* da transação para o *input* da nova proposta. Nestas situações, há uma verificação adicional, para testar se o estado de referência é o último estado atualizado. Se tudo correr corretamente, cria-se a sequência de transações. Desta forma esta arquitetura reforça a integridade da informação, e o nível de segurança das transações [32].

3.2.3. Contratos

Relembra-se que uma transação só será aceite exclusivamente se todos os seus assinantes aprovarem a(s) proposta(s). Quando é verificada a validade de uma transação, averigua-se também se os seus *inputs* e *outputs* não sofrem do duplo gasto. Uma transação será contratualmente válida se todos os seus *inputs* e *outputs* forem aceites, de acordo com o contrato em questão. Na imagem abaixo, podemos observar melhor como é feita a verificação, quanto à validade de um contrato.

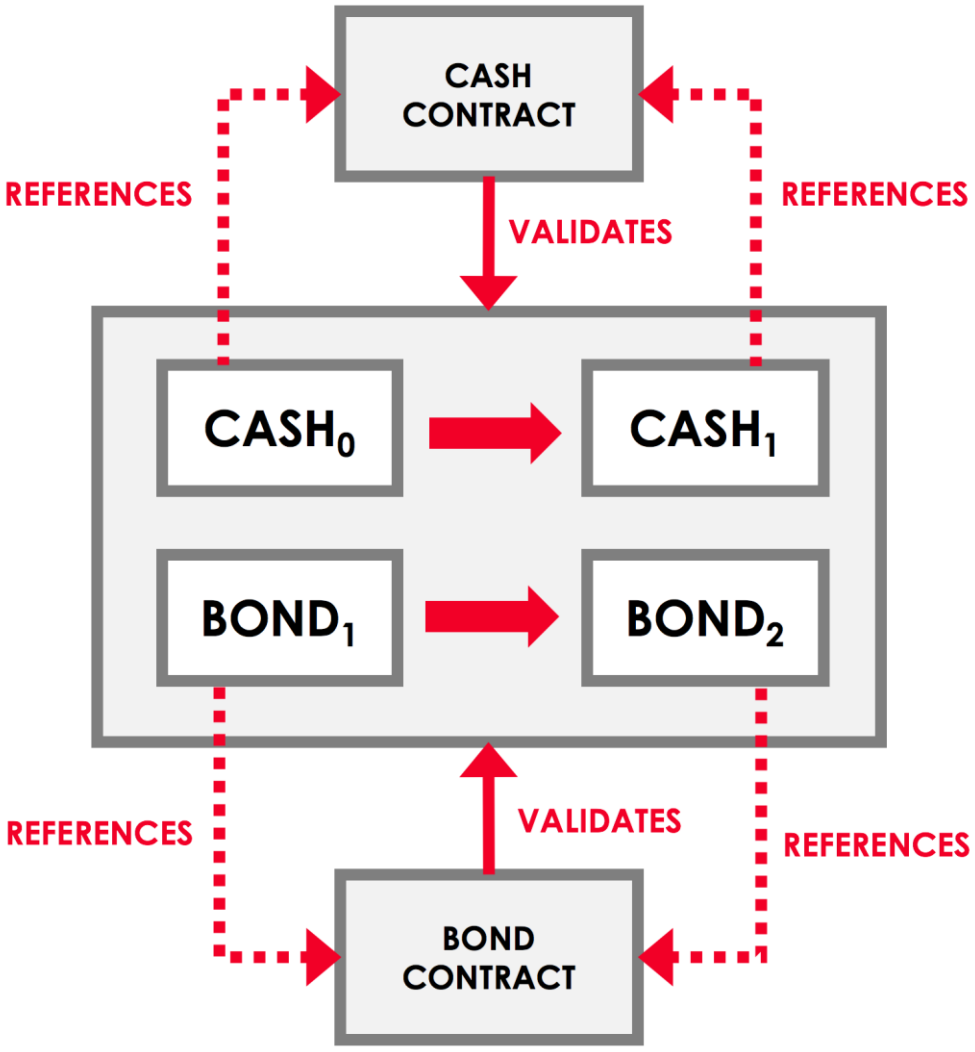

Figura 3.7. – Exemplo de como um contrato valida pedidos de transação. Retirado de [33].

Sendo a transação o link transitório entre o *input* e *output*, o mesmo vai verificar a sua validade ao contrato em questão. No caso da transação de $CASH_0$ para $CASH_1$ tanto o *input* $CASH_0$ como *output* $CASH_1$ vão referenciar a validade dos seus termos e condições ao contrato. O contrato tem acesso a todos os dados de *inputs*, *outputs*, e outras informações pertencentes às propostas.

Todas as propostas de uma transação têm de obter aprovação nos seus contratos. A transação só é validada e aprovada para ser o novo estado atualizado nos registos, apenas quando todas as propostas forem válidas. Se, pelo menos, uma proposta não for validada por um contrato, toda a transação será recusada [33],[34].

3.2.4. Fluxos

Como a rede *Corda* utiliza comunicação ponto-a-ponto, e não uma comunicação de *Broadcast* [35], onde uma mensagem é enviada para todos os nós na rede, surge a necessidade dos nós especificarem, com exatidão, a informação que é preciso enviar, para quem, e por que ordem.

A necessidade de especificar todo o processo de comunicação e transação deram origem às *flow frameworks*, ou fluxos. As *flow frameworks* são sequências de passos automatizados e programados, com antecedência, para dizer ao nó como alcançar o estado atualizado. Por outras palavras, é uma sequência de passos que a rede *Corda* irá fazer, sempre que existir necessidade de uma troca entre nós.

As *flow frameworks* são totalmente customizáveis ao desejo da entidade que as cria. São também visíveis para todos os membros da rede, garantindo integridade, e um sistema aberto para com os nós participantes.

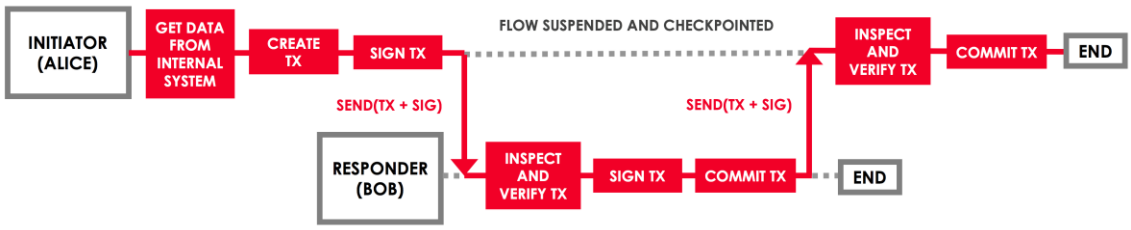

Figura 3.8. – Exemplo de uma *flow framework* entre dois nós. Retirado de [32].

Na Figura 3.8, podemos analisar o exemplo de uma simples *flow*, em que o objetivo da mesma é assegurar a comunicação entre Alice e Bob. Nesta *flow*, optou-se por decidir que se Alice quer comunicar com Bob, terá de se mostrar como online no sistema. Para isso, terá que fazer a sua verificação de identidade no sistema interno. Depois, poderá criar a sua proposta de transação, e, antes de ser enviada a Bob, terá de a assinar, garantindo, assim, que Bob recebe uma proposta imodificável – uma vez assinada e enviada, a proposta é definitiva. Esta é, assim, enviada para Bob, que verifica a sua validade. Se passar os testes é, então, assinada, e aprovada a sua atualização nos registos. A transação é enviada, de volta, para Alice, já com a assinatura de Bob, que, por sua vez, verifica se os dados de Bob são os corretos, válidos. Se a resposta a esta questão for positiva, aprova, por sua vez, a atualização da mesma para os registos. Uma vez aprovada, é criado o estado da transação. Este é apenas um exemplo de uma transação simples, com um *flow* simples. No *Corda* as *flows* são feitas pelas entidades que criam os seus serviços, e podem ser customizáveis, como o seu criador quiser.

Qualquer nó pode ter um ou mais *flows*, por cada serviço ativo que tenha. Todos são completamente automatizados e disponíveis sempre que o nó escolher [36]. Para facilitar o desenvolvimento, o *Corda* fornece uma biblioteca de *flows*, com tarefas básicas e comuns para os desenvolvedores terem mais facilidade e rapidez na produção dos mesmos [34].

3.2.5. Consenso

O consenso é o processo que consiste em verificar se a transação é aceite por todos os contratos para todos os inputs e outputs e se todas as transações têm todas as assinaturas corretamente. Estas verificações são feitas para a proposta atual e para todas as transações na sequência de transações que participaram para obter a proposta atual. Ou seja, não é suficiente verificar e validar a proposta atual. É necessário verificar e revalidar todas as transações anteriores que deram origem ao input da proposta atual. Podemos ver um exemplo na Figura 3.9 [37],[38].

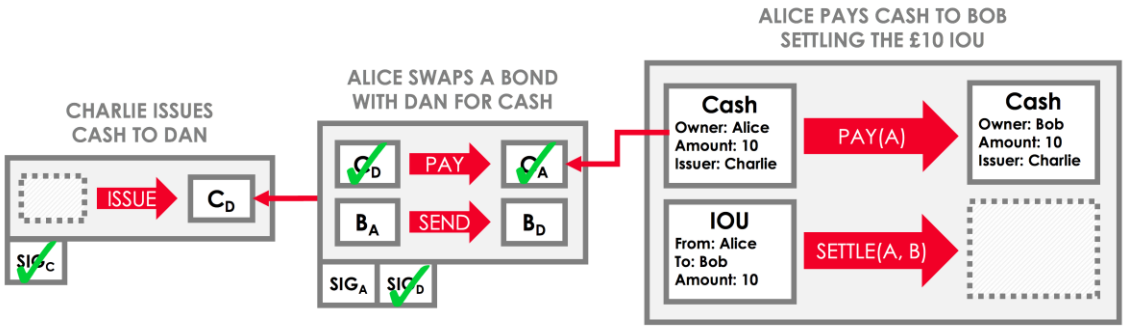

Figura 3.9. – Exemplo da verificação do consenso entre dois nós. Retirado de [37].

Na Figura 3.9, temos o exemplo de como é feita a verificação do consenso. A sequência começa com Charlie, que fornece dinheiro a Dan. De seguida, Dan faz uma troca com Alice, na qual lhe dá material, a troca de dinheiro. E, no fim, temos a proposta atual, na qual Alice solda uma dívida de 10 libras com Bob. Sucede que Bob não participou nas transações anteriores, o que significa que não tem registado, na sua base de dados, as sequências necessárias para verificar que as 10 libras de Alice são válidas. Isto posto, Bob, antes de assinar a proposta, vai requisitar a Alice a sequência de transações referentes às suas 10 libras. Alice, a partir da sua base de dados, envia a Bob a sequência de transações, e se Bob validar, assina e envia de volta à Alice, aceitando as suas 10 libras. Se tudo correr devidamente, passa-se a ter consenso entre as duas entidades, e, consequentemente, um estado atualizado.

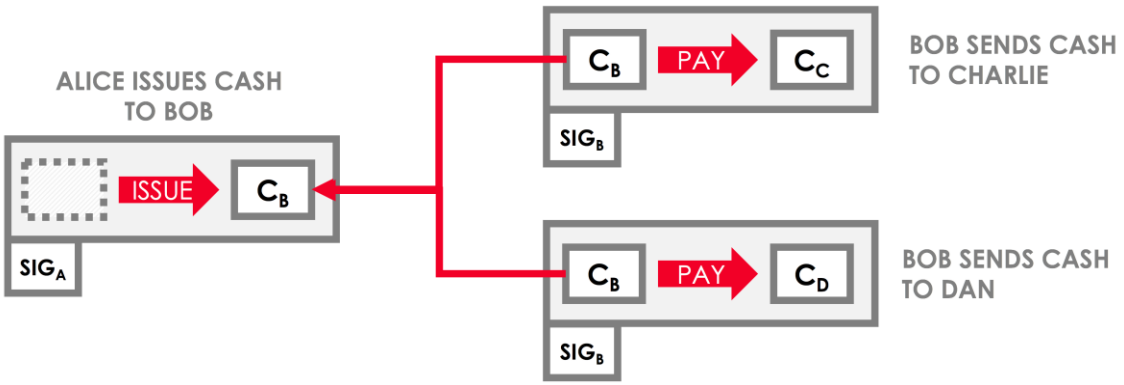

Figura 3.10. – Exemplo do problema do duplo gasto. Retirado de [37].

Na Figura 3.10, podemos considerar como a unicidade do consenso é necessária. No exemplo, temos Alice, que fornece a Bob 1 000 000 de libras. Bob decide fazer duas propostas: na primeira proposta, Bob quer transferir 1 000 000 de libras para Charlie, em troca de 900 000 de libras; na segunda proposta, Bob quer transferir 1 000 000 de libras para Dan, em troca de 800 000 de libras. Isto vai gerar a questão do duplo gasto, pois Bob vai conseguir gastar, ou trocar, os seus bens mais que uma vez, e ganhar mais no processo. É igualmente um problema, porque a transação vai ser apenas válida quando for efetuada a confirmação de que Bob tem, efetivamente, na sua posse, 1 000 000 de libras. Dada essa situação, o Corda impôs a regra da unicidade do consenso, que consiste na exclusividade dos inputs, referentes a cada proposta, não podendo ser consumidos ou usados noutras propostas. Devido a essa regra, Bob não conseguiria validar a sua proposta, e, com isso, esta seria refutada antes de sequer ter a chance de alcançar Charlie ou Dan. Desta forma, resolve-se o problema do duplo gasto [37].

3.2.6. Notários

O notário é um serviço da rede *Corda*, que fornece verificações para a unicidade das transações, e propostas entre os nós. Quando um nó pede a assinatura do notário, este assegura-se que a mesma proposta é válida, quanto à sua unicidade de consenso, e, se for, assina e envia-a de volta para o nó. Se a proposta não passar a verificação, vai ser rejeitada e assinalada, como tentativa de duplo gasto. O nó, ao receber a informação sobre a possibilidade de a proposta ser um duplo gasto, descarta-a por completo.

Todas as propostas na rede *Corda* passam pelos nós notários, por obrigatoriedade da arquitetura do mesmo, o que providencia segurança quanto à certidão e precisão das propostas, quando vindas do notário.

O *Corda* fornece vários serviços de notariado, sendo a diferença entre estes o nível de privacidade, a escalabilidade, os sistemas legais, a compatibilidade, e a agilidade algorítmica. Por norma, há vários nós notários por cada rede *Corda,* reduzindo-se a latência nas transações, o congestionamento da rede e a privacidade, dados os vários nós com diferentes serviços [39],[40].

4. Desenvolvimento

O plano para este capítulo é a implementação de um modelo, que comprove a funcionalidade, a integridade, a privacidade e a segurança por trás do *Corda*, no que poderia ser trocas de valores entre clientes e empresas.

O *Corda* simplificou bastante o processo de desenvolvimento de *CorDapps*, de forma a que os desenvolvedores estejam mais focados no desenvolvimento de serviços através do mesmo, e não no seu funcionamento. O processo foi simplificado, ao ponto de oferecer um *template* de testes para o desenvolvimento de serviços. No *template*, apenas existe a necessidade de desenvolver o comportamento dos registos, contratos, *frameworks*, criar os nós na rede para efetuar pedidos, e um notariado para verificar e validar. O funcionamento dos nós, os métodos básicos de transferência, comunicação e informação na rede são todos fornecidos pelo *Corda*.

4.1. Definição do modelo

As trocas, comunicações, entre todo o resto, que acontecem entre nós, na rede *Corda*, são determinadas pelas *frameworks* por trás dos serviços propostos. Determinar o modelo da *framework* é o equivalente a fazer o modelo de um serviço, tal como o seu funcionamento. Para descrever o serviço que pretendo implementar, irei fazê-lo através da *framework* apresentada na Figura 4.1[36].

O nó emissor será um dos três nós a ser criado para esta rede, e o mesmo para o nó recetor. Para o notário, é apenas necessário criar a sua instância, visto que este é fornecido pelo *Corda*. O notário fará a verificação da unicidade de consenso, autonomamente, e, consequentemente, determinará se o pedido de transação é valido nesse aspeto, ou não, prevenindo duplos gastos.

A construção do pedido será feita numa classe à parte, na classe dos registos. Existe toda a flexibilidade para o nível de complexidade de como se pretende que o pedido seja feito, e do que é guardado nos registos finais. Para dados de teste será um construtor simples, em que, tanto o emissor como o recetor, terão anonimato total, relativamente à sua localização, e nomes de utilizador. Tanto um como o outro terão nomes fictícios na rede, mascarados por Party A, Party B ou Party C. No registo, será guardado apenas o nome fictício do emissor, do recetor, e a quantia emitida de um para o outro.

A verificação do pedido será feita pela classe do contrato, não esquecendo que são os contratos que verificam se um pedido é contratualmente válido. A verificação será feita antes do pedido ser sequer assinado, e enviado para o notário. O *Corda* permite total flexibilidade na

quantidade de parâmetros que o contrato pode verificar, para aceitar ou recusar um pedido. Para este modelo, a verificação será feita em vários parâmetros explicados mais à frente.

O nó recetor verifica o pedido. A verificação também será feita com a classe do Contrato, mas agora do lado do nó recetor. É importante fazer a distinção, pois há que fazer a verificação de que a entidade que recebe o valor emitido é, efetivamente, a entidade certa, etc. Uma vez válido, o nó recetor assina o pedido, e, consequentemente o pedido é registado. O mesmo se sucede para o emissor.

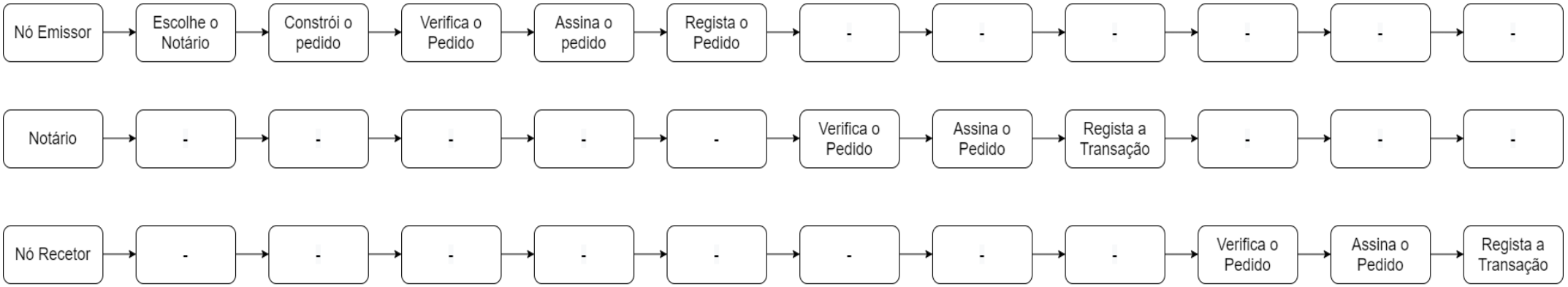

Figura 4.1. - *Framework* que descreve o funcionamento do serviço.

4.2. Código

O código divide-se em quatro partes: a primeira parte é dedicada ao construtor dos factos, nos registos; a segunda parte é relativa aos contratos, e quais são os parâmetros que o mesmo verifica; a terceira parte é a automação de todo o processo através da *framework* implementada, e explicada na Figura 4.1. A quarta e última parte é dedicada à criação dos nós de teste, e notariado para a simulação do modelo.

4.2.1. Registos (*TokenState*)

Para fazer a construção dos factos nos registos, a classe *TokenState* foi criada, para guardar o nome do emissor, o nome do recetor e a quantia que circulou entre eles. O código consiste num simples construtor com *gets*, para cada parâmetro. No fim, a lista é guardada com os nomes, ou identificadores de quem participou na transação. É possível observar o código no Anexo 7.1.

Tabela 4.1. - Descrição dos métodos utilizados para criar o código do *TokenState*.

Métodos	**Descrição**
Party	Classe que serve de construtor para a identificação dos nós. Guarda parâmetros como a chave pública e nome real ou fictício.
List<AbstractParty> getParticipants	Classe que serve de construtor para os nós, e retorna os participantes da transação. Pode guardar as sequências de transações de cada participante na transação, o seu hash e/ou chaves públicas.
getIssuer	Serve para retornar o emissor da transação.
getOwner	Serve para retornar o recetor da transação.
getAmount	Serve para retornar o valor da transação.

4.2.2. Contrato (*TokenContract*)

Relembra-se que o contrato verifica se o pedido é contratualmente válido, e que o *Corda* fornece uma grande flexibilidade sobre a quantidade de parâmetros que podemos verificar. Para o contrato efetuam-se as seguintes verificações: se o pedido tem ou não *Inputs* – só serão aceites pedidos sem *Inputs* prévios, ou seja, sem antes terem sido gastos os valores que estarão a ser enviados e sem que tenham sido utilizados noutras transações e/ou pedidos. Desta forma qualquer pedido é único. Os valores têm de ser enviados para algum destino, mas não a mais que um sítio e/ou recetor. É aceite apenas um comando, ou seja, apenas aceitamos um tipo de serviço (envio de unidades monetárias) para dados de simplicidade de testes. Revendo-se que todos os pedidos

e transações estão conectados aos contratos, neste caso, verifica-se se o pedido em causa é coerente com o formato de pedido da classe *TokenState*. O valor emitido tem de ser positivo, e o comando ser do tipo *Issue* (emitir). Por fim verifica-se se o emissor assinou o pedido, antes de o enviar. É possível observar o código no Anexo 7.2.

Tabela 4.2. - Descrição dos métodos utilizados para criar o código do *TokenContract*.

Métodos	Descrição
Party	Classe para guardar a identificação do emissor.
List<PublicKey>	Classe utilizada para guardar as chaves públicas dos nós participantes na transação.
LedgerTransaction tx	Apontador para o último pedido não verificado no registo.
tx.getCommand(0)	Command é um vetor de comandos a espera de serem iniciados. Neste caso só aceito um comando de cada vez, então vou à posição zero do vetor verificar se o pedido é do tipo *Issue.*
tx.getOutput(0)	Retorna a instância para o tipo de pedido.

4.2.3. Framework (*TokenIssueFlowInitiator*)

Para automatizar todo o modelo mostrado na Figura 4.1, criou-se a classe *TokenIssueFlowInitiator*. Funciona como o nível inferior do modelo em que recebe os dados, e envia para os outros níveis – *TokenState*, Notário, *TokenContract*, emissor e recetor. O código é visível no Anexo 7.3.

O código, do Anexo 7.3, segue a ordem do fluxograma da Figura 4.1. Começa-se por invocar um notário, e guardar a identidade do criador do pedido. De seguida, constrói-se o pedido e acrescenta-se-lhe o comando de *Issue*. Envia-se o pedido ao notário, para este o verificar, e de seguida, é enviada para o recetor. Este verifica a validade contratual do pedido, e, uma vez aprovado, o pedido é assinado e registado. Após ser registado, o pedido é enviado para o emissor, e este regista-o, igualmente, tornando-se, assim, o pedido numa transação.

Tabela 4.3. - Descrição dos métodos utilizados para criar o código do *TokenIssueFlowInitiator*.

Métodos	Descrição
Party	Classe que serve de construtor para a identificação dos nós. Guarda parâmetros como a chave pública e nome real ou fictício.
TransactionBuilder	Construtor que guarda uma variedade de pedidos e verificações a serem verificados e/ou testados.
FlowSession	Serve para estabelecer a comunicação entre um nó e o seu *flow*.
SignedTransaction	É o nível mais alto da hierarquia de cada nó. Contém um construtor com as assinaturas dos nós participantes juntamente com vários comandos que podemos executar para obter mais funções e/ou métodos de proteção.

4.2.4. Participantes

Para simular o modelo, foram criados três nós e um notário. Para isso, basta correr o comando fornecido pela documentação do *Corda*, e o mesmo cria três nós e um notário com dados aleatórios [41]. Desta forma, simula uma rede local composta por quatro elementos, e, neste caso, a conFiguração dos três nós e notário pode ser vista no Anexo 7.4.

4.2.5. Simulação

Como o Corda fornece uma interface, em formato de *Secure Shell* (SSH), podemos simular a *CorDapp*, sem a necessidade de desenvolver interfaces adicionais. Dado que que os nós são iniciados com o comando *build/nodes*, é simulada uma rede, composta pelos quatro elementos anteriormente mencionados [10]. Os nós estão identificados como Party A, Party B, e Party C, na Figura 4.2, e de seguida, o notário como *Notary*, na Figura 4.3.

Figura 4.2. - Interface dos 3 nós criados.

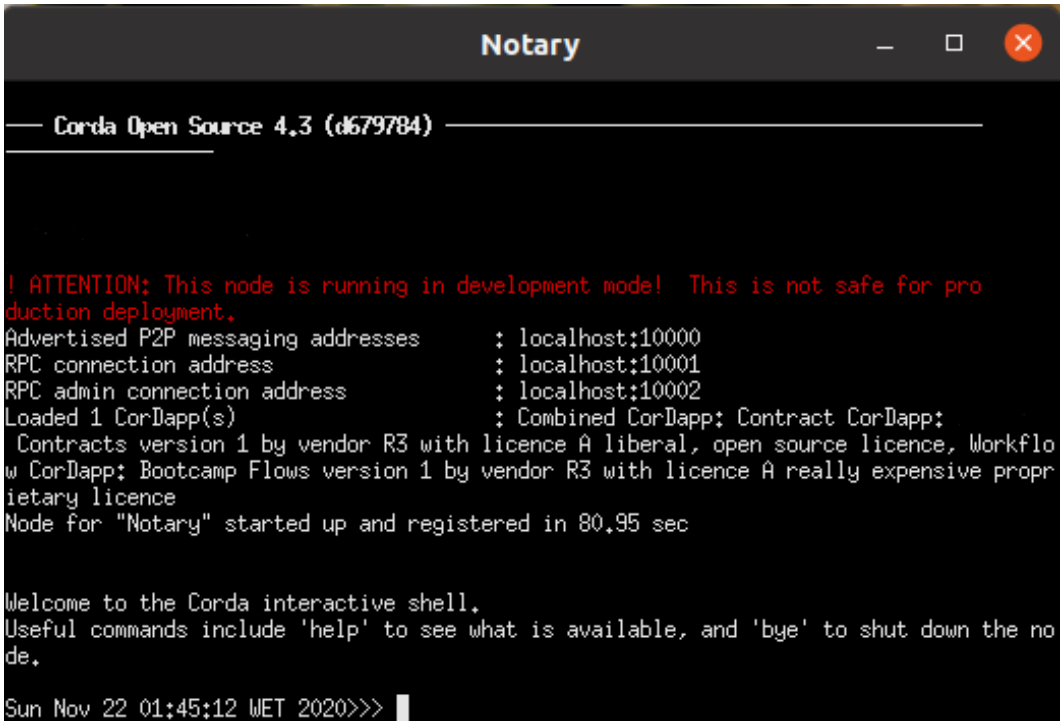

Figura 4.3. - Interface do notário.

O objetivo é passar valor de um para outro nó. Ou seja, passar uma determinada quantia de *tokens*, neste caso, para dados de teste, 99 unidades monetárias do Party A para o Party B. O teste será positivo se os valores passarem e, ao verificar o repositório de transações no nó que não participou, confirmar que este não tem conhecimento da transação que ocorreu entre os outros dois.

No nó A (Party A) começa-se por verificar se o modelo do fluxograma funciona com o comando "*flow start TokenIssueFlowInitiator"*, mas sem lhe atribuir parâmetros, como é possível observar na Figura 4.4.

Figura 4.4. - Demonstração de como o pedido é recusado se não tiver o formato correto.

Como esperado, é recebida uma mensagem de erro, visto inicializar um pedido, sem o formato desenvolvido, no *TokenState*. De seguida, volta-se a inicializar o serviço, procurando bugs, alternando os dados em falta e, como suposto, aparece uma mensagem de erro, sinalizando os

parâmetros em falta, e, consequentemente, mostrando a estrutura correta para a iniciação do pedido. Seguido de mais uns testes de inserção de dados incorretos, prossegue-se com o formato finalmente correto, como é possível observar na Figura 4.5:

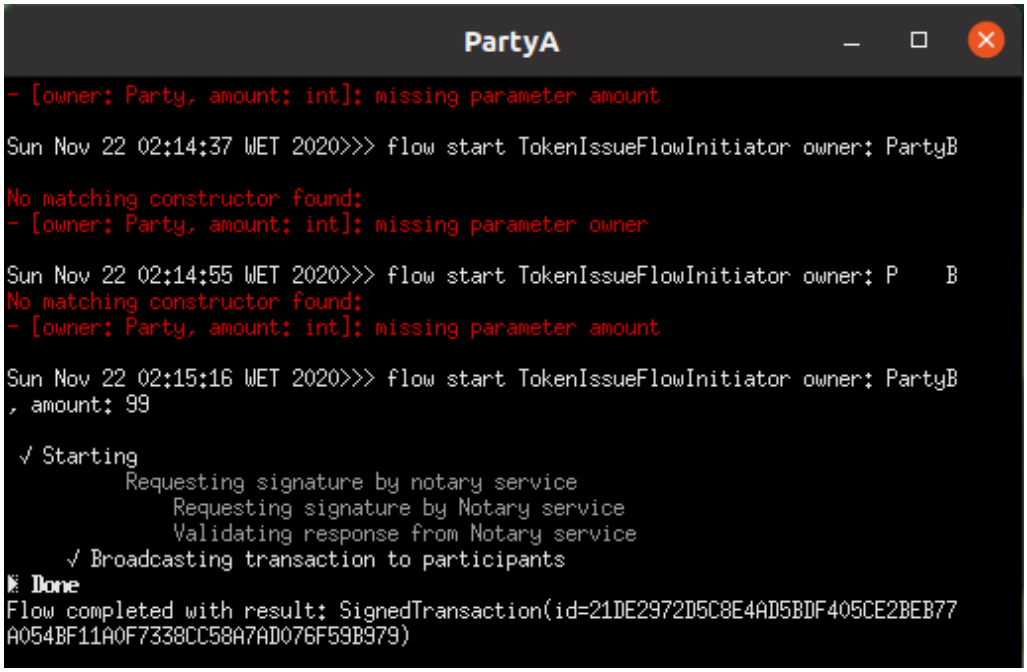

Figura 4.5. - Pedido criado, registado e verificado pelo notário quanto à integridade de informação. Também é enviado ao resto dos participantes da transação.

É feita uma transferência do Party A para o Party B, de 99 unidades monetárias. Recebe-se as mensagens de texto, iguais ao modelo descrito, anteriormente na Figura 4.1. O emissor inicializa um pedido, sendo este contratualmente verificado e validado, e, de seguida, enviado para o notário, obtendo a sua assinatura e aprovação. Por fim, o pedido é enviado aos restantes participantes da transação, que neste caso sendo só o Party B.

Para fazer uma verificação adicional, retifica-se, na base de dados do Party B, se os valores foram recebidos. Na Figura 4.6, é possível observar os resultados dessa verificação, observando-se que o Party B tem os dados da transação, inclusive os dados do emissor, recetor, notário, quantia transferida e o *hash* da transação, o que significa que o Party B recebeu o pedido, verificou-o contratualmente, tendo sido aprovado, e consequentemente, assinado. Como tem todas as condições necessárias, o Party B assinou o pedido e registou-o, seguindo-se da transação de sucesso entre os dois nós.

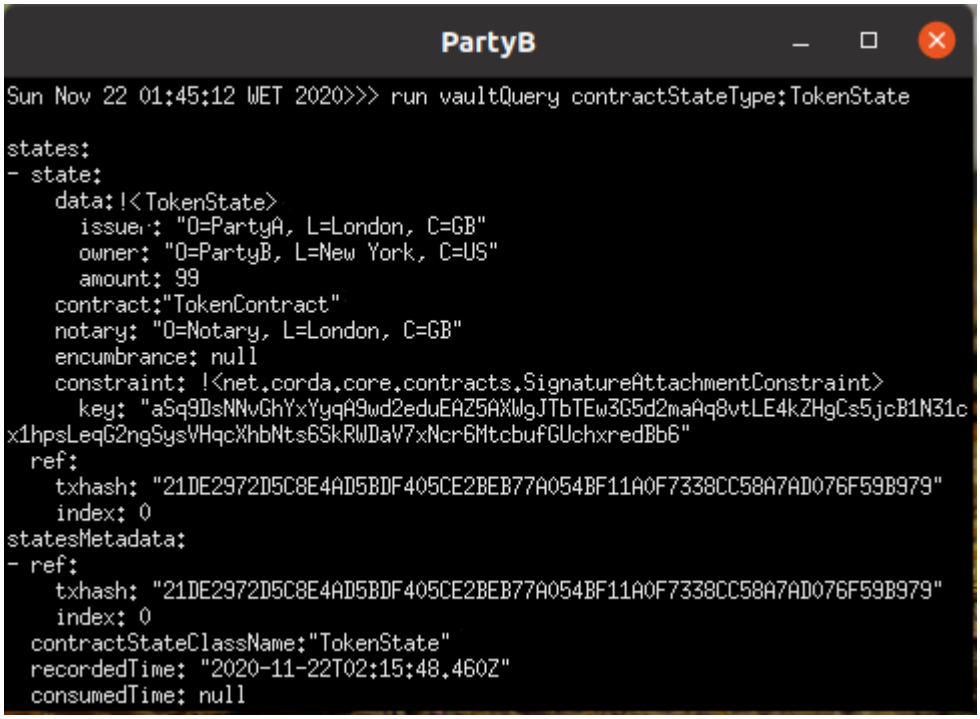

Figura 4.6. – Verificação do conteúdo dos estados do Party B.

Fazendo a mesma verificação para o Party A, Figura 4.7:

Figura 4.7. – Verificação do conteúdo dos estados do Party A.

Por fim, dirige-se o Party C, que é o nó que não participou na transação. Confirma-se que ele não tem dados, nem sabe da existência da transação entre o Party A e Party B. Para isso, inicia-se o comando de verificação da base de dados do Party C, e verifica-se que este não tem quaisquer registos na sua base de dados, como é possível observar na Figura 4.8:

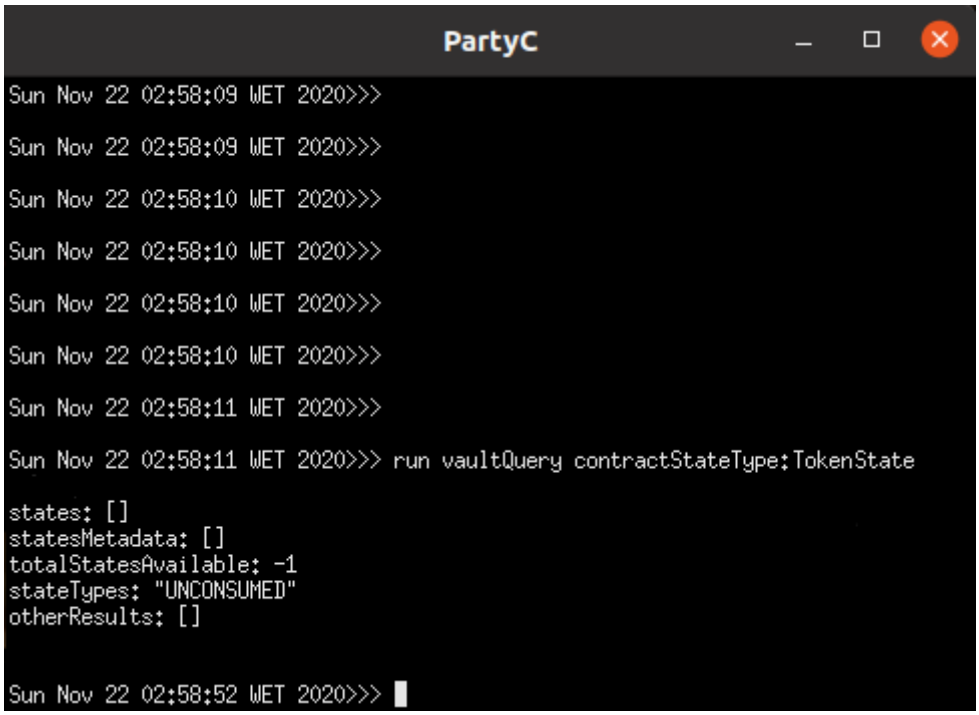

Figura 4.8. – Verificação do conteúdo dos estados do Party C.

4.2.6. Discussão de Resultados

Todos os testes correram como suposto. A transação entre dois nós correu com os devidos parâmetros, e com os esperados meios de segurança e prevenção. Obteve-se garantia de que a transação foi verificada contratualmente, e também pela unicidade de consenso fornecida pelo notário. Isto fornece a tão necessária e essencial integridade nas trocas de informação, valores, ativos, etc. Aumenta-se, também, o nível de segurança de transações, e criação de pedidos/serviços na rede. Só depois destas verificações é que o pedido é assinado e enviado para os outros participantes – neste caso Party B. Este voltou a verificar, contratualmente, a validade do pedido, e também recebeu a assinatura do notário, garantindo a segurança quanto à validade e integridade da informação que recebeu, e prosseguir, sem preocupações, com a transação. Para acrescentar, o terceiro nó, Party C, que não participou na transação, não tem qualquer informação sobre o que os outros dois nós trocaram. Servindo de prova para a segurança, e a total privacidade para com as entidades exteriores, nas trocas entre participantes.

É de notar que o *Corda* permite uma enorme flexibilidade em como tudo é feito, desde a criação dos pedidos até as assinaturas dos recetores. Com um simples sistema, conseguimos comprovar que o *Corda* é a escolha certa para o objetivo desta tese: é encontrar uma plataforma, capaz de fornecer segurança, integridade, serviço contínuo e privacidade.

5. Conclusão e Perspetivas Futuras

Há grandes problemas no mundo financeiro, relativamente à segurança no transporte de informação, e, sobretudo, à sua integridade. Esta tese foca-se na fraude financeira, e, sobretudo, na falha de integridade que existe – e continuará a existir – no sistema atual. Com esta tese, o objetivo foi encontrar uma plataforma que servisse de alternativa, e corrigisse os problemas existentes no sistema financeiro do mundo real. Entre três plataformas fidedignas, mas, no entanto, diferentes entre si, o *Corda* foi o eleito, para prosseguir com o desenvolvimento do objetivo. As razões por trás da escolha são, resumidamente, a extrema flexibilidade da tecnologia em questão, a sua facilidade de integrar e permitir uma segurança rígida, e privacidade ao nível da escolha do utilizador.

O excelente desenvolvimento pela equipa do *Corda* conseguiu alcançar um desenvolvimento fácil e simples, para desenvolvedores externos. É apenas necessário modificar os registos, contratos, e fluxos para conseguir uma *CorDapp*. Partindo desse princípio, foi elaborado um fluxograma, com os estados a seguir, na prova de conceito da aplicação do *Corda*.

Os testes relativos à simulação tiveram cem por cento de coerência e certidão para com as previsões no modelo, pelo que se tornou possível concluir que o *Corda* é um excelente candidato para o mundo financeiro. O seu nível de segurança flexível, e as suas lógicas de tráfego rápido permitirão transações e serviços com altos níveis de privacidade, segurança, transparência e integridade.

Apesar do teste ser suficiente para comprovar o conceito do *Corda* ser a plataforma certa, certamente haverão funções e serviços que poderiam ser também desenvolvidos.

O *Corda* permite altos níveis de complexidade referentemente aos seus serviços. Dito tal, o trabalho desta tese poderia ser elevado, mais à frente, com o desenvolvimento de serviços adicionais, nomeadamente uma interface em HTTP, a que um utilizador pudesse aceder e fazer simulações de condições de seguro, sendo o prémio pelo mesmo variável, conforme as opções escolhidas pelo utilizador.

Também é possível realizar estudos, e explorar as funcionalidades mais complexas do *Corda*, designadamente funcionalidades como *Time-Windows*, em que uma transação só pode ser validada durante um período limitado, sendo o notário a entidade a recusar a transação, caso esta se encontre fora do período acessível [42]. *Oracles*, que servem para tornar as transações mais flexíveis, permitindo-lhes adicionar condições extra [43]. Um exemplo prático seria um contrato de seguro, em que o cliente deseja adicionar novos termos e condições ao seu contrato, explorando

novas opções, valores, etc. *Transaction tear-offs*, em que certos componentes de uma transação podem ser escondidos de alguns participantes, por motivos de segurança e/ou privacidade [32]. Exemplificando, quando é fornecido um código PIN a um cliente, que só este o deve conhecer, e surge a necessidade de esconder essa informação de todos, exceto do cliente em questão. Estes, e muitos outros componentes que o *Corda* fornece, ao serem explorados, permitirão o desenvolvimento de aplicações, com um grau de complexidade muito mais elevado. Certamente permitiria explorar mais os limites e capacidades do *Corda*, e fornecer uma opinião mais realista sobre o quanto mais poderia elevar a qualidade dos serviços existentes no mundo real.

6. Referências

[1] I. de S. de Portugal, Ed., *Contrato de Seguro*, 2011th ed. Etigrafe, Lda., 2011.

[2] I. Gorjão Garcia and Maria do Rosário Costa Silva Veiga, "O COLAPSO DO GRUPO BANCO ESPÍRITO SANTO: ANÁLISE DOS RELATÓRIOS E CONTAS E IDENTIFICAÇÃO DE 'BANDEIRAS VERMELHAS,'" ISCTE, Dissertação de Mestrado, Setembro de 2017.

[3] M. Gazzoni, "Seguradora é condenada por falsificar assinatura de cliente em contrato," *G1-Economia*, 2018.

[4] L. Adriano, "State Farm lawsuits claim agents faked signatures," *Insurance Business America*, 2018.

[5] R. Z. Wiggins, T. Piontek, and A. Metrick, "Overview," *Yale Progr. Financ. Stab.*, pp. 1–23, 2014.

[6] E. Santo, "The Fall of the House of Espírito Santo," *Markets/Finance*, 2014.

[7] W. I. S. Blockchain, "MAKING SENSE OUT OF BLOCKCHAIN TECHNOLOGIES.", https://mackinstitute.wharton.upenn.edu/wp-content/uploads/2018/10/Blockchain_Strategic-Path_White-Paper.pdf, [Accessed: November-2020]

[8] I. Nath, "Data Exchange Platform to Fight Insurance Fraud on Blockchain," *IEEE Int. Conf. Data Min. Work. ICDMW*, vol. 0, pp. 821–825, 2016, doi: 10.1109/ICDMW.2016.0121.[9] M. Mittal, "Using Blockchain Smart Contracts to Drive Service Excellence in Insurance Sector," *The Journal of Insurance Institute of India*, pp. 28–31, 2018.

[10] S. Kim et al., "A Feature based Content Analysis of Blockchain Platforms," *Int. Conf. Ubiquitous Futur. Networks, ICUFN*, vol. 2018-July, pp. 791–793, 2018, doi: 10.1109/ICUFN.2018.8436843.

[11] Bitcoin.org, "Bitcoin - Open source P2P money," *Bitcoin.org/en*, 2014. [Online]. Available: https://bitcoin.org/en/. [Accessed: 12-Jan-2020]

[12] HYPERLEDGER, "Whitepaper Introduction Hyperledger.", *The Linux Foundation*, pp.1-33, 2016.

[13] A. Shanker, "Public vs. private blockchains," *PC Mag.*, pp. 114–115, 2017.

[14] Corda, "Corda: The open source blockchain for business," 2019. [Online]. Available: https://www.corda.net. [Accessed: 12-Jan-2020]

[15] R. G. Brown, "The Corda Platform: An Introduction," *Corda Platf.*, pp. 1–21, 2018.

[16] M. C. and B. Liskov, "Practical Byzantine Fault Tolerance," *Laboratory for Computer Science, Massachusetts Institute of Technology*, 1999.

[17] M. Hearn, R. G. Brown, "Corda: A distributed ledger," *Whitepaper*, pp. 1–73, 2019.

[18] "R3 Corda Master documentation," 2015. [Online]. Available: https://docs.corda.net/.

[19] The Linux Foundation, "Supporting Members – Hyperledger," 2020. [Online]. Available: https://www.hyperledger.org/members.

[20] IBM, "Hyperledger Fabric – Hyperledger," *The Linux Foundation*. [Online]. Available: https://www.hyperledger.org/projects/fabric.

[21] A. Konnov, A. Makarov, M. Pozdnyakova, R. Safin, and A. Salagaev, "Russia," *Juv. Delinq. Eur. Beyond Results Second Int. Self-Report Delinq. Study*, no. February, pp. 359–368, 2010, doi: 10.1007/978-0-387-95982-5_25.

[22] Ethereum Foundation, "Home | Ethereum," *Ethereum.Org*, 2014. [Online]. Available: https://ethereum.org/.

[23] V. Buterin, “A next-generation smart contract and decentralized application platform,” *Etherum*, no. January, pp. 1–36, 2014.

[24] “Coinbase Pro | Digital Asset Exchange.” [Online]. Available: https://pro.coinbase.com/.

[25] R3 Corda LTD, “Network | Corda OS 4.5 | Corda Documentation,” *R3 LTD*, 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-ecosystem.html. [Accessed: 13-Sep-2020].

[26] Y. Iwamoto, “A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications,” *Nippon rinsho. Japanese J. Clin. Med.*, vol. 68 Suppl 9, pp. 7–12, 2010.

[27] Oz Barak, Assaf Touboul, “Point to point link and communication method,” U.S. Patent No. (7,593,729B2), no. 12, 2008.

[28] R. Oppliger, “Security at the internet layer*,” Computer (Long. Beach. Calif).*, vol. 31, no. 9, pp. 43–47, 1998, doi: 10.1109/2.708449.

[29] R3 Ltd 2020, “Ledger | Corda OS 4.5 | Corda Documentation,” Feb-2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-ledger.html. [Accessed: 18-Nov-2020].

[30] H. J. Fowler and W. E. Leland, “Local Area Network Traffic Characteristics, with Implications for Broadband Network Congestion Management,” *IEEE J. Sel. Areas Commun.*, vol. 9, no. 7, pp. 1139–1149, 1991, doi: 10.1109/49.103559.

[31] R3 Corda LTD, “States | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-states.html.

[32] R. C. LTD, “Transaction tear-offs | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-tearoffs.html.

[33] R. C. LTD, “Contracts | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-contracts.html. [Accessed: 24-Nov-2020].

[34] R. C. LTD, “Contract catalogue | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/contract-catalogue.html. [Accessed: 24-Nov-2020].

[35] G. C. Ware and F. Llp, “Secure broadcast system and method,” vol. 1, no. 19, 2003.

[36] R. C. LTD, “Flows | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-flows.html. [Accessed: 24-Nov-2020].

[37] R. C. LTD, “Consensus | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-consensus.html.

[38] M. Valenta and P. Sandner, “Comparison of Ethereum, Hyperledger Fabric and Corda,” *Frankfurt Sch. Blockchain Cent.*, no. June, p. 8, 2017.

[39] D. Mohanty, R3 Corda for Architects and Developers. 2019.

[40] R. C. LTD, “Notaries | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-notaries.html. [Accessed: 24-Nov-2020].

[41] R. C. LTD, “Creating nodes locally | Corda OS 4.5 | Corda Documentation,” 2019. [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/generating-a-node.html. [Accessed: 22-Nov-2020].

[42] R. C. LTD, “Time-windows | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-time-windows.html.

[43] R. C. LTD, “Oracles | Corda OS 4.5 | Corda Documentation.” [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-oracles.html.

[44] Halas, Milan, “Blockchain technology as part of Distribution system Operators”, *School of business and Management*, pp. 43-70, 2019

7. Anexos

```java
@BelongsToContract(TokenContract.class)
public class TokenState implements ContractState{

    private Party issuer;
    private Party owner;
    private int amount;

    public TokenState(Party issuer, Party owner, int amount) {
        this.issuer = issuer;
        this.owner = owner;
        this.amount = amount;
    }

    //get methods
    public Party getIssuer() { return issuer; }
    public Party getOwner() { return owner; }
    public int getAmount () { return amount; }

    //lista de participantes na transação
    public List<AbstractParty> getParticipants() {
        List<AbstractParty> participants = new ArrayList<>();
        participants.add(issuer);
        participants.add(owner);
        return participants;
    }
}
```

Anexo 7.1. – Código da classe *TokenState*.

```java
public class TokenContract implements Contract {


    @Override
    public void verify(LedgerTransaction tx) throws IllegalArgumentException {

        if (tx.getInputStates().size() != 0)
            throw new IllegalArgumentException("Não pode haver Inputs.");
        if (tx.getOutputStates().size() != 1)
            throw new IllegalArgumentException("Tem que haver pelo menos um Output.");
        if (tx.getCommands().size() != 1)
            throw new IllegalArgumentException("Não pode ter mais que um comando.");

        ContractState output = tx.getOutput( index: 0);
        if(!(output instanceof TokenState))
            throw new IllegalArgumentException("Tem que ser instância de TokenState.");

        TokenState token = (TokenState) output;
        if (token.getAmount() <= 0)
            throw new IllegalArgumentException("O valor emitido tem que ser positivo.");

        //tipo de pedido
        Command command = tx.getCommand( index: 0);
        if (!(command.getValue() instanceof Commands.Issue))
            throw new IllegalArgumentException("O comando tem de ser do tipo Issue.");

        //lista de assinaturas
        List<PublicKey> requiredSigners = command.getSigners();
        Party issuer = token.getIssuer();
        PublicKey issuerKey = issuer.getOwningKey();
        if(!(requiredSigners.contains(issuerKey)))
            throw new IllegalArgumentException("O emissor 'Issuer' tem que ser um assinante.");
    }


    public interface Commands extends CommandData {
        class Issue implements Commands { }
    }
}
```

Anexo 7.2. – Código da classe *TokenContract*.

```java
@InitiatingFlow
@StartableByRPC
public class TokenIssueFlowInitiator extends FlowLogic<SignedTransaction> {
    // Participantes neste serviço
    private final Party owner;
    private final int amount;

    // constructor
    public TokenIssueFlowInitiator(Party owner, int amount) {
        this.owner = owner;
        this.amount = amount;
    }

    // made bby Corda
    private final ProgressTracker progressTracker = new ProgressTracker();

    @Override
    public ProgressTracker getProgressTracker() { return progressTracker; }

    //Start flow
    @Suspendable
    @Override
    public SignedTransaction call() throws FlowException {
        // We choose our transaction's notary (the notary prevents double-spends).
        Party notary = getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);

        // Guardo a identidade do emissor
        Party issuer = getOurIdentity();

        // Criamos o pedido TokenState.
        TokenState tokenState = new TokenState(issuer, owner, amount);
        TokenContract.Commands.Issue command = new TokenContract.Commands.Issue();

        // Prepara-se o pedido para ser enviado e verificado pelo notário.
        TransactionBuilder transactionBuilder = new TransactionBuilder();
        transactionBuilder.setNotary(notary);
        transactionBuilder.addOutputState(tokenState, TokenContract.ID);
        transactionBuilder.addCommand(command, issuer.getOwningKey(), owner.getOwningKey());

        // Verifica-se a validade contratual do pedido.
        transactionBuilder.verify(getServiceHub());

        FlowSession session = initiateFlow(owner);

        // Assina-se o pedido para o tornar imutavel
        SignedTransaction signedTransaction = getServiceHub().signInitialTransaction(transactionBuilder);

        // The counterparty signs the transaction
        SignedTransaction fullySignedTransaction = subFlow(new CollectSignaturesFlow(signedTransaction, singletonList(session)));

        // We get the transaction notarised and recorded automatically by the platform.
        return subFlow(new FinalityFlow(fullySignedTransaction, singletonList(session)));
    }
}
```

Anexo 7.3. – Código da classe *TokenIssueFlowInitiator*.

```
task deployNodes(type: net.corda.plugins.Cordform, dependsOn: ['jar']) {
    //Java version check
    if (JavaVersion.current() != JavaVersion.VERSION_1_8){
        throw new GradleException("This build must be run with java 8")
    }
    nodeDefaults {
        projectCordapp { deploy = true }
    }
    node {
        name "O=Notary,L=London,C=GB"
        notary = [validating: false]
        p2pPort 10000
        cordapps = []
        rpcSettings {
            address("localhost:10001")
            adminAddress("localhost:10002")
        }
    }
    node {
        name "O=PartyA,L=London,C=GB"
        p2pPort 10003
        rpcSettings {
            address("localhost:10004")
            adminAddress("localhost:10005")
        }
        rpcUsers = [[user: "user1", password: "test", permissions: ["ALL"]]]
    }
    node {
        name "O=PartyB,L=New York,C=US"
        p2pPort 10006
        rpcSettings {
            address("localhost:10007")
            adminAddress("localhost:10008")
        }
        rpcUsers = [[user: "user1", password: "test", permissions: ["ALL"]]]
    }
    node {
        name "O=PartyC,L=Lagos,C=NG"
        p2pPort 10009
        rpcSettings {
            address("localhost:10010")
            adminAddress("localhost:10011")
        }
        rpcUsers = [[user: "user1", password: "test", permissions: ["ALL"]]]
    }
}
```

Anexo 7.4. – Configuração dos 3 nós e notário na rede.